МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Секція інформаційно-комунікаційних технологій

ВИПУСКНА РОБОТА

на тему:

«Додаток для миттєвого обміну повідомленнями»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Власенко О.В.**

Студентки групи ІН – 62 **Левченко Т.В.**

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2020 р.

ЗАВДАННЯ

до випускної роботи

Студентки четвертого курсу, групи ІН-62 спеціальності "Інформатика" денної форми навчання Левченко Тетяни Володимирівни.

Тема: "**Додаток для миттєвого обміну повідомленнями**"

Затверджена наказом по СумДУ

№_____ від _____2020 р.

Зміст пояснювальної записки: 1) постановка завдання; 2) інформаційний огляд; 3) опис основних технологій реалізації 4) розробка інформаційного й програмного забезпечення інформаційної системи; 5) аналіз результатів моделювання.

Дата видачі завдання "_____" _____2020 р.

Керівник випускної роботи _____ Власенко О.В.

Завдання приняв до виконання _____ Левченко Т.В.

РЕФЕРАТ

Записка: 74 стор., 32 рис., 13 табл., 1 додаток, 21 джерело.

Об'єкт дослідження – додаток для миттєвого обміну повідомленнями – месенджер «Secret Messenger».

Мета роботи – розробка desktop-додатку для миттєвого обміну повідомленнями між користувачами мережі Інтернет, забезпечивши повне шифрування даних.

Методи дослідження – в ході виконання проекту було використано мову програмування Java, брокер повідомлень RabbitMQ та СУБД MySQL.

Результати — розроблено месенжер для миттєвого обміну повідомленнями у мережі Інтернет, в якому забезпечено шифрування повідомлень за допомогою алгоритму RSA. Передбачена можливість використання власного серверу для зберігання повідомлень.

МЕСЕНДЖЕР, ОБМІН ПОВІДОМЛЕННЯМИ, ШИФРУВАННЯ ДАНИХ, БРОКЕР ПОВІДОМЛЕНЬ, АЛГОРИТМ RSA

ЗМІСТ

ВСТУП

У даний час інформаційні технології стрімко розвиваються кожну секунду нашого життя. Технологічний, а також інформаційний прогрес, зробив величезний крок вперед. Сучасне людство не представляє своє життя без гаджетів.

“Миттєві повідомлення або, повніше, система обміну миттєвими повідомленнями – телекомунікаційна служба для обміну текстовими повідомленнями між комп'ютерами або іншими пристроями користувачів через комп'ютерні мережі” [1].

Головна відмінність миттєвих повідомлень у месенджері від листів на електронну пошту в тому, що обмін першими відбувається в реальному часі, миттєво [1]. Найчастіше користувачу приходить сповіщення про повідомлення, а тому його буде складніше загубити. При відправлені повідомлення електронною поштою воно зберігається у поштовій скриньці на сервері. Для того, щоб побачити повідомлення, отримувач повинен сам перевірити свою поштову скриньку на наявність нового листа і відповісти на отримані повідомлення. У інтернет-месенджерах зв'язок між користувачами тримається постійно, неперервно і відправлене повідомлення одразу передається користувачу-отримувачу [1]. Саме тому цей спосіб спілкування є дуже популярним.

Месенджери ще називають клієнтами, тому що вони підключаються до центрального серверу, а не працюють окремо та самостійно. Тобто, система миттєвих повідомлень найчастіше використовує серверні протоколи для підтримки своєї роботи. Сучасні програми повинні бути безпечними, високопродуктивними, нейтральними до архітектури. Створюється рішення, яке увібрало б всі ці основні характеристики, а основною особливістю була б саме безпека обміну повідомленнями. Тому дана робота є актуальною в даний момент.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Огляд існуючих додатків для обміну повідомленнями

Все більше людей надають перевагу спілкуванню в месенджерах, ніж розмові по телефону. Темп життя різко пришвидшився з початком використання новітніх технологій та Інтернету. Зараз навіть не обов'язково набирати текст, можна записати голосове повідомлення та відправити його у месенджер. Саме тому вони так швидко підкорюють нову аудиторію та стрімко розвиваються.

Існує велика кількість месенджерів, їх популярність варіюється в залежності від країни. Виконаємо огляд існуючих додатків, які є широко використовуваними в Україні.

За даними експертів найпопулярнішим месенджером для користувачів в Україні є Viber. Він посів першу позицію виходячи із такої цифри як середньодобова доля використань серед всіх месенджерів, яка дорівнює 87%. На другому місці Facebook Messenger, середньодобова доля якого становить 48%. Третє місце посідає Telegram, який відрізняється своєю швидкозростаючою популярністю [2].

1.1.1 Месенджер Viber

Viber перш за все був призначений для текстових повідомлень, його головною функцією можна вважати саме їх передачу. Текст можна редагувати і видаляти (з самого початку таких функцій не було, вони додалися з оновленнями). Однак текст повідомлення не можна робити жирним, курсивом або закресленим. Існує можливість відправляти тимчасові повідомлення, які будуть доступні тільки протягом встановленого користувачем часу. Також в чат можна додавати GIF-анімацію, відео, голосові повідомлення, геолокацію. Крім того, в Viber доступні відео- та аудіодзвінки. Мессенджер дозволяє здійснювати дзвінки і на звичайні номери, проте ця функція надається за додаткову плату [3].

Найчастіше користувачі встановлюють саме мобільну версію цього додатку для спілкування в Інтернеті. Нижче наведено зображення дизайну чатів у Viber (рис.1.1).

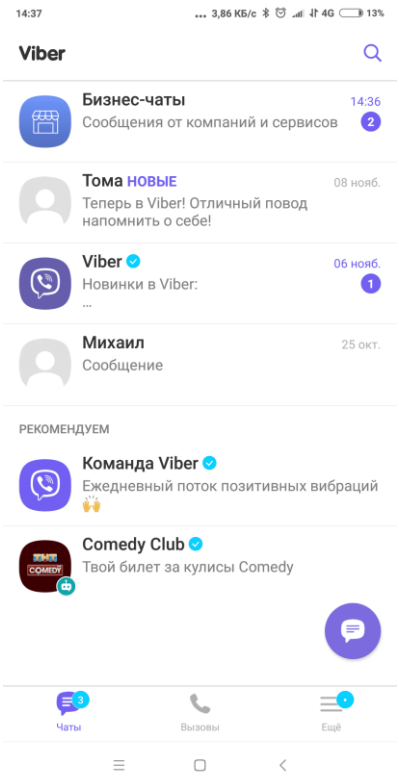

Рисунок 1.1 – Дизайн додатку Viber

Більшу частину заходів, які використовує Viber для захисту даних, можна назвати стандартними. Так, наприклад, в месенджері застосовується SSL-шифрування каналу зв'язку. Крім того, в ньому використовується алгоритм SHA-256RSA для сертифікатів підтвердження.

Для користувачів передбачені і додаткові заходи захисту. До них відноситься верифікація контактів. Ця функція дозволяє синхронізувати ключі зі співрозмовником. При підозрі в тому, що чиїмось контактом заволоділи треті особи, можна запросити порівняння ключа, тобто перевірити верифікацію. Якщо ключі не співпадуть, такого користувача можна буде заблокувати.

Крім того, існують і інші способи підвищити безпеку в месенджері. По-перше, це секретні чати. У них працює end-to-end шифрування, яке дозволяє приховати вміст листування від третіх осіб. При цьому, провайдер стверджує, що

ці дані будуть доступні тільки на пристроях співрозмовників, однак, незважаючи на це, пропонує завантажити їх і на desktop-версію. По-друге, як уже було сказано, в Viber є приховані чати. Їх вміст не так захищено, як в секретних чатах, однак їх буде складніше знайти тій людині, яка отримає доступ до пристрою. Щоб відкрити їх, потрібен пін-код.

В угоді користувача Viber відзначається, що персональну інформацію користувача можуть розкрити не тільки за запитом суду і органів нагляду, а й для тих третіх осіб, які можуть допомагати здійснювати діяльність сервісу. При цьому, всіх перерахованих вище третіх осіб ніяк не обмежують в використанні цих даних. Все це є явним недоліком у політиці безпеки месенджера.

Інший підозрілий момент пов'язаний із секретними чатами. Їх суть полягає в тому, що дані з такого листування не зберігаються на сервері провайдера, а доступні виключно на пристроях співрозмовників. Однак сервіс пропонує отримати до них доступ і з третього пристрою. Це означає, що інформація все ж може залишати межі секретного чату, а, при бажанні, її можуть використовувати провайдер або навіть інші особи [3]. Деталей, як саме надається доступ з інших пристроїв, немає.

1.1.2 Messenger від Facebook

До стандартних для додатку функцій можна віднести наступні. У першу чергу, він дозволяє обмінюватися миттєвими текстовими повідомленнями. Їх можна редагувати протягом певної кількості часу, а також видаляти, в тому числі – у всіх співрозмовників.

До повідомлення можна прикріплювати файли різного формату, геолокацію і моментальні фотографії або відео, які можна зняти спеціально для месенджера. Сервіс підтримує можливість запису голосових повідомлень, а також аудіо- та відео-дзвінки.

Ряд функцій спрямований на роботу з чатами. Наприклад, можна здійснювати пошук в листуванні. Також можна створювати групи з декількома

учасниками і секретні чати [4]. Нижче приведено зображення вікна зі звичайними чатами (рис.1.2) та вікно секретного чату з користувачем (рис.1.3).

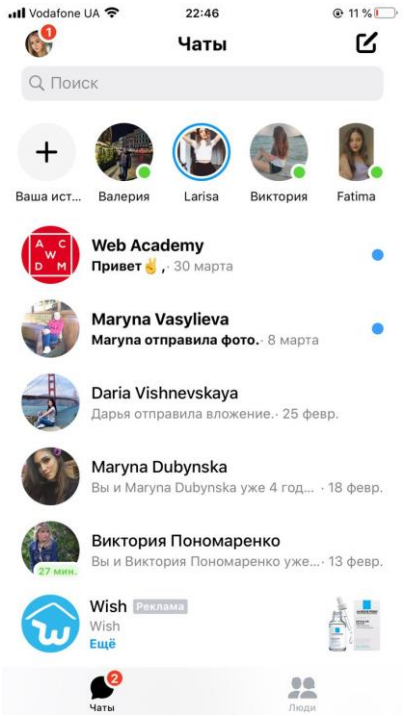

Рисунок 1.2 – Вікно чатів у Facebook Messenger

Рисунок 1.3 – Секретний чат у Facebook Messenger

Facebook Messenger має декілька ступенів захисту. З одного боку, його веб-версія використовує безпечний протокол HTTPS, а сама програма працює на базі мережевого протоколу MQTT. Крім того, для групових чатів застосовується протокол шифрування Asynchronous Ratcheting Tree (ART), який дозволяє створювати ключі не тільки для активних учасників листування, а й для тих, хто знаходиться оффлайн. З іншого боку, можна активувати додатковий ступінь захисту, не передбачений за замовчуванням.

Під додатковим захистом маються на увазі секретні чати, де застосовується криптографічний протокол Signal, що поєднує в собі такі елементи, як 3-DH, AES-256, HMAC-SHA256. Завдяки цьому протоколу стає можливим end-to-end шифрування, при якому листування можуть бачити тільки відправник і одержувач [4]. Для збільшення безпеки в секретних чатах можна налаштувати

видалення повідомлень через певний проміжок часу. У такому випадку листування ніде не збережеться.

Основною проблемою, пов'язаною з безпекою цього сервісу, є часті повідомлення про витік даних декількох мільйонів користувачів Facebook. За словами представників компанії, ці прецеденти пов'язані з атаками хакерів і стали можливі завдяки існуючим вразливостям. Вони також запевняють, що усунули всі помилки і постійно посилюють заходи безпеки. Однак після подібних випадків довіряти сервісу стає складно. Немає гарантії того, що сам Facebook може бути неопосередковано причасний до витоків конфіденційної інформації.

1.1.3 Месенджер Telegram

Основною ідеєю сервісу є безпечний обмін повідомленнями. Саме це нещодавно призвело до конфлікту з владою кількох країн і подальшого блокування сервісу. Telegram виконує основне завдання усіх месенджерів – миттєву передачу повідомлень. Користувач може не тільки відправляти повідомлення, пересилати, цитувати їх, але і вибирати додаткові налаштування. Наприклад, існує функція відкладених повідомлень, завдяки якій можна вибрати дату і час їх доставки співрозмовнику, а також прихованих чатів. Крім того, повідомлення можна редагувати і навіть видаляти. Причому, це відображається не тільки у відправника, а й у одержувача.

Крім текстових послань, телеграм дозволяє записувати голосові і відео-повідомлення, а також здійснювати аудіо-дзвінки. При цьому, відео-дзвінки в телеграм ще не передбачені.

Месенджер надає можливість додавання ярликів на робочий стіл, що приєднані до певного аккаунту користувача. Всі важливі повідомлення для зручності можна відправляти в "Обране" – по суті, листування з самим собою. У телеграм існують не тільки приватні чати. В месенджері є канали та групи, де можуть одночасно можуть бути присутні багато користувачів [5].

Нижче приведено зображення того, як виглядає список чатів (рис.1.4) та яку інформацію містить сам чат (рис.1.5).

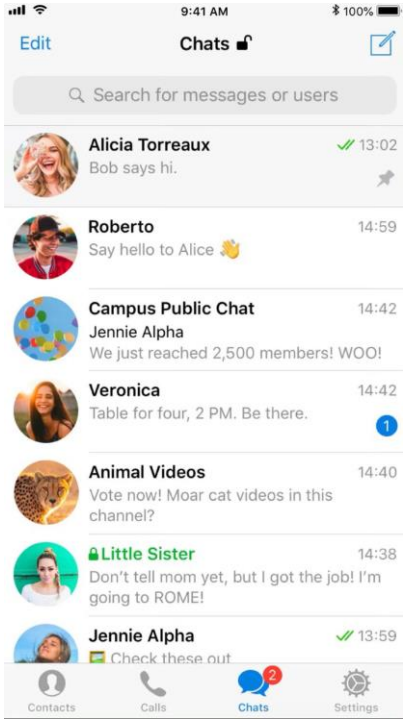

Рисунок 1.4 – Дизайн списку чатів Telegram

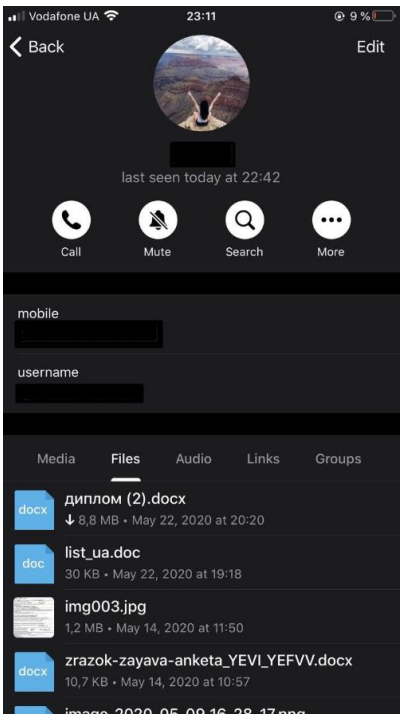

Рисунок 1.5 – Інформація про чат у Telegram

Зробивши акцент на безпечності передачі повідомлень, розробники телеграм використовували з цією метою відразу декілька методів захисту. У загальних випадках для шифрування застосовуються такі алгоритми, як RSA-2048, DH-2048, SHA-256 і AES-256, а для веб-версії – протокол HTTPS. Крім того, месенджер став тестовим майданчиком для самостійної розробки брата Павла Дурова – криптографічного протоколу MTProto. Так, стало можливим створення секретних чатів, в яких застосовується end-to-end шифрування, завдяки якому повідомлення зберігаються тільки на пристроях відправника і одержувача і не залишаються на сервері [5]. Іншими словами, в цьому випадку у сторонніх немає доступу до листування.

Безпека забезпечується в тому числі непрямими способами. Наприклад, в телеграм є можливість пошуку користувача, номер телефону якого невідомий. Замість нього застосовується нік (логін) людини в месенджері.

Виходячи з вищенаведеної інформації можна зробити порівняльну таблицю (табл. 1.1) розглянутих месенджерів та зробити висновки про функції та задачі, які є основними для виконання роботи.

Таблиця 1.1 – Порівняльна таблиця месенджерів

Назва	Facebook Messenger	Viber	Telegram
Алгоритм шифрування	AES-256, HMAC-SHA256, 3-DH	SHA-256RSA	RSA-2048, DH-2048, AES-256, SHA-256
Додаткові методи захисту	HTTPS, MQTT	SSL	HTTPS
Секретні чати	+	+	+
Аудіо-дзвінки	+	+	+
Відео-дзвінки	+	+	–

Проаналізувавши функції, переваги та недоліки кожного з месенджерів по основним критеріям, можна описати, якими функціями має бути наділений мій месенджер. Перш за все, він повинен здійснювати обмін миттєвими повідомленнями між користувачами. Це буде здійснюватися у чатах, які за замовчуванням будуть секретними. Версія додатку 1.0 повинна містити основні функції для користувачів, тому реалізація голосових повідомлень, аудіо- та відео-дзвінків буде відкладено до майбутніх версій. Також потрібно обрати набір алгоритмів для шифрування даних.

1.2 Огляд існуючих мов програмування, що підходять для створення месенджеру

Для розробки месенджеру має бути обрана одна із мов програмування, яка може бути використана для розробки Backend, а також за допомогою якої можна зробити не веб-додаток, а саме desktop-додаток.

Одним із найвикористовуваніших рейтингів мов програмування є показник TIOBE Index. За допомогою нього можна перевірити наскільки популярна та чи інша мова програмування, відповідно і кількість фахівців, літератури присвяченій тій чи іншій мові, кодів з відкритим доступом тощо. Нижче приведене зображення рейтингу популярності мов (ТОП-10) у порівнянні між квітнем 2020-го року та аналогічного періоду за 2019-й рік (рис. 1.6).

Apr 2020	Apr 2019	Change	Programming Language	Ratings	Change
1	1		Java	16.73%	+1.69%
2	2		C	16.72%	+2.64%
3	4	^	Python	9.31%	+1.15%
4	3	v	C++	6.78%	-2.06%
5	6	^	C#	4.74%	+1.23%
6	5	v	Visual Basic	4.72%	-1.07%
7	7		JavaScript	2.38%	-0.12%
8	9	^	PHP	2.37%	+0.13%
9	8	v	SQL	2.17%	-0.10%
10	16	^^	R	1.54%	+0.35%

Рисунок 1.6 – Рейтинг популярності мов програмування

Перш за все розглянемо мову програмування Java, адже на сьогодні вона займає перше місце у рейтингу. Це дуже відома мова, здатна працювати на будь-якій платформі. Розробники створили віртуальну машину Java (JVM), яка дала першу мову програмування з гаслом: «Написати один раз, запустити де завгодно». Більш того, зараз більшість операційних систем просто зобов'язані включати її до свого складу, так як робота деяких додатків без цього компілятора буде недостатньо результативною.

Виділимо 3 основні характеристики:

- Java пропонує потужну, багатофункціональну парадигму, інтерпретовану мову програмування з помірною кривою навчання та високою продуктивністю для розробників;

- Java знатно відстала, що є ключовою вимогою для бізнес-програм. Java ніколи не вводила таких значних змін, як Python або Scala. Як результат, це все ще вибір номер один для підприємств, адже Java – це стабільність.
- час виконання Java JVM – шедевр інженерії програмного забезпечення та одна з найкращих віртуальних машин у цій галузі. Завдяки 25-річній інноваційній та інженерній майстерності, JVM пропонує високу продуктивність та можливості Java. Також JVM надає Java передовий Garbage Collection [6].

Однак є у мови і підводні камені. Так, програми, написані на Java, мають репутацію занадто повільних і вимагають великих обсягів оперативної пам'яті. Java має кілька модифікацій (НЕ діалектів, а саме різних видів), які створені для різних цілей. Кожен з них має свої унікальні бібліотеки даних і структуру, що дозволяє більш продуктивно працювати над певним напрямком програмування. [7]

В останні роки Java втратила частину своїх прихильників через сучасні мови, які дуже зручні для розробників, особливо Python, JavaScript.

Проте, Java працює над своїми недоліками і намагається стати придатною для найбільш сучасних додатків. Java все ще є мовою програмування номер один для підприємств.

Python – останнім часом дуже популярна мова програмування. Її головна ідея полягає у спрощенні всього, що можна спростити. Тому ця мова користується популярністю у новачків, для деяких програмістів Python є першою в їх арсеналі. Розробники роблять все для того, щоб якомога простіше було створювати складні програми. Ця мова програмування створена на основі більш ранніх мов та ввібрала в себе всі їхні напрацювання, переваги і вдосконалила їх. До основних плюсів її компілятора можна віднести: мінімалізм, багатофункціональність та простоту. Але в свою чергу, за такий мінімалізм доводиться миритися з низькою швидкодією, а також наявністю багатьох помилок в системному коді, деякі з яких навіть присутні і в останніх версіях [10].

Три основні характеристики:

- USP Python – це його мовна конструкція. Це високопродуктивна, елегантна, проста, але потужна мова програмування. Python встановив золотий стандарт з точки зору досвіду розробників;
- Python має першокласну інтеграцію з C / C ++ і може легко завантажувати важкі завдання процесора на C / C ++. Також Python надає потужний набір інструментів для математики, статистики та обчислювальної техніки з різними бібліотеками. Як результат, Python домінує в таких сферах, як Machine Learning / Deep Learning / Data Science та інших наукових областях;
- у Python є дуже активна спільнота та підтримка. Ви завжди можете знайти достатню кількість бібліотек і фреймворків Python, незалежно від того, працюєте ви над Enterprise Applications, Data Science або Artificial Intelligence [9].

За останні кілька років в Python спостерігається величезне зростання популярності, без ознак сповільнення.

Далі можна розглянути таку мову програмування, як C#. Дана мова використовує об'єктно-орієнтований підхід до програмування у всьому. Це означає, що потрібно описувати абстрактні конструкції на основі предметної області, а потім реалізовувати між ними взаємодію. Даний підхід користується великою популярністю.

Сьогодні C # – це багатопрограмна мова програмування, яка широко використовується не тільки на платформі Windows, але і на платформі iOS / Android (завдяки Xamarin) та платформах Linux [8].

Розглянемо 3 основні характеристики мови:

- Андерс Хейльсберг зробив чудову роботу, щоб вивести C # з тіні Java та надати власну ідентичність. Щодо досвіду розробника, C # випереджає Java.
- підтримуючи Microsoft і перебуваючи в галузі протягом 20 років, C # має великі екосистеми бібліотек та каркасів. ASP.NET використовується для веб-розробки, особливо в Windows.
- як і Java, C # також не залежить від платформи (завдяки CLR) і працює на пристроях Windows, Linux, Mobile [6].

1.3 Огляд основних баз даних

Бази даних – це спеціально розроблене сховище для різних типів даних. Кожна база даних, має певну модель (реляційна, документно-орієнтована), яка забезпечує зручний доступ до даних. Системи управління базами даних (СКБД) – спеціальні додатки (або бібліотеки) для управління базами даних різних розмірів і форм.

Реляційна СУБД повинна забезпечувати реляційну модель роботи з даними. Сама модель має на увазі певний тип зв'язку між сутностями з різних таблиць. Щоб зберігати і працювати з даними, такий тип СУБД повинен мати певну структуру (таблиці). У таблицях кожен стовпець може містити дані різного типу. Кожен запис складається з безлічі атрибутів (стовпців) і має унікальний ключ, що зберігається в тій же таблиці – всі ці дані взаємопов'язані між собою, як описано в реляційної моделі [9].

Три найрозповсюджених СУБД, а саме:

- SQLite – дуже потужна система управління;
- MySQL – найпопулярніша і поширена СУБД;
- PostgreSQL – найбільш продвинута СУБД.

СУБД SQLite легко вбудовується в додаток. Ця система надає розробнику широкий вибір інструментів, для роботи та використання СКБД. Вона базується на файлах, тому при роботі з SQLite звернення відбуваються безпосередньо до цих файлів (в них зберігаються дані про базу даних), замість портів і гнізд в мережевих СУБД. SQLite швидка та потужна завдяки технологіям своїх бібліотек.

Переваги SQLite:

- як і вказувалось вище, файлова структура SQLite дає велику перевагу над усіма іншими СКБД. Уся база даних складається з одного файлу, тому має таку характепристику як «легка переносимість»;

- може здатися, що ця СУБД примітивна, але вона використовує SQL. Деякі особливості відкинуті (RIGHT OUTER JOIN або FOR EACH STATEMENT), але основні все-таки підтримуються.

Недоліки SQLite:

- відсутність системи користувачів – більші СУБД включають в себе системи управління правами доступу користувачів. В більшості випадків відсутність цієї функції не є критичною, так як ця СУБД використовується в невеликих додатках (як і в нашому випадку, це не є визначальним);
- виходячи з проектування, реалізації SQLite досить складно досягнути високої продуктивності з нею [9].

Далі розглянемо MySQL. Це найпоширеніша повноцінна серверна СУБД. MySQL дуже функціональна та вільно розповсюджувана СУБД, яка успішно працює з різними сайтами і веб-додатками. Незважаючи на те, що в ній не реалізований весь SQL функціонал, MySQL пропонує досить багато інструментів для розробки додатків. Так як це серверна СУБД, додатки для доступу до даних, на відміну від SQLite працюють зі службами MySQL.

Переваги MySQL:

- ця СКБД простота в роботі. Встановити MySQL досить просто. Також існують різноманітні програми, наприклад, GUI (графічний інтерфейс), що дозволяє досить легко працювати з СКБД;
- MySQL має багатий функціонал та підтримує більшість функцій стандатного SQL;
- MySQL має багато функцій у своєму арсеналі, що забезпечують безпеку, окрім того, більшість з них підтримується за замовчуванням;
- СКБД працює з великими обсягами даних без проблем і легко масштабується, це гарний вибір для великих додатків;
- спрощення деяких стандартів дозволяє MySQL значно збільшити свою продуктивність, відповідно і додатку.

Недоліки MySQL:

- MySQL має певні проблеми з надійністю. Через деякі способи обробки даних (наприклад, зв'язки, транзакції, аудити) ця СКБД іноді поступається іншим по показнику надійності [9].

PostgreSQL є найбільш професійною з усіх трьох вище розглянутих СУБД. Вона максимально відповідає стандартам SQL. Від інших СУБД PostgreSQL відрізняється підтримкою затребуваного об'єктно-орієнтованого і / або реляційного підходу до баз даних. Наприклад, повна підтримка надійних транзакцій, тобто атомарность, послідовність, ізоляційні, міцність (Atomicity, Consistency, Isolation, Durability (ACID).) Завдяки потужним технологіям Postgre дуже продуктивна. Паралельність досягнута завдяки реалізації управління різноманітним паралелізмом (MVCC), що також забезпечує відповідність підходу ACID. PostgreSQL можна легко розширювати власними (збереженими) процедурами. Ці функції спрощують використання постійно повторюваних операцій.

Переваги PostgreSQL:

- відкрите ПЗ відповідає стандарту SQL. PostgreSQL – безкоштовне ПЗ з відкритим вихідним кодом. Ця СУБД є дуже потужною системою;
- велика спільнота. Існує досить велика спільнота, де запросто можна знайти відповіді на питання;
- велика кількість доповнень. Незважаючи на величезну кількість вбудованих функцій, існує дуже багато доповнень, що дозволяють розробляти дані для цієї СУБД і управляти ними;
- розширення – існує можливість розширення функціоналу за рахунок збереження своїх процедур.

Недоліки PostgreSQL:

- при простих операціях читання PostgreSQL може значно уповільнити сервер і бути повільніше своїх конкурентів, наприклад, MySQL;
- популярністю ця СУБД похвалитися не може, хоча і є досить велика спільнота;

- хостинг – в силу названих вище чинників іноді досить складно знайти хостинг з підтримкою цієї СУБД [9].

1.4 Постановка завдання

Метою випускної роботи є розробка додатку для миттєвого обміну повідомленнями – месенджеру з високим рівнем захисту даних користувачів. Додаток матиме такі основні функції: вибір серверу для користування, пошук користувача за логіном та обмін повідомленнями з користувачами.

Для розробки додатку потрібно виконати такі задачі:

- Провести аналіз основних алгоритмів шифрування та запровадити безпечний алгоритм для шифрування у додатку;
- Розробити месенджер на мові програмування Java з усіма необхідними функціями для миттєвого обміну повідомленнями.

Месенджер є достатньо актуальним для застосування в реальному житті, а простота його майбутнього інтерфейсу дозволить користувачам швидко розібратися та почати використовувати його.

За головну мету ставиться саме захист інформації, серверна частина додатку, а не розробка дизайну.

2. ВИБІР МЕТОДУ РІШЕННЯ

2.1 Мова програмування

Для реалізації додатку було обрала мову програмування Java. Це об'єктно-орієнтована мова, зручна і надійна в експлуатації завдяки таким своїм перевагам, як багатозадачність, підтримка протоколів Internet і мультиплатформенність.

Java включає в себе об'єктно-орієнтоване програмування (ООП) – концепцію, в якій ви не тільки визначаєте тип даних і його структуру, а й набір функцій, що застосовуються до нього. Таким чином, структура даних стає об'єктом, яким можна управляти для створення відносин між різними об'єктами.

Можна виділити такі плюси використання ООП:

- при ООП можна повторно використовувати об'єкти в інших програмах;
- ООП попереджує помилки, оскільки об'єкти приховують інформацію, до якої не повинно бути доступу;
- ООП більш ефективно організовує структуру програм;
- ООП спрощує обслуговування і модернізацію старого коду.

Java не так доброзичлива до новачків, як Python, проте досить проста для будь-якого розробника з базовим розумінням фреймворків, пакетів, класів і об'єктів. Це проста, типізована і передбачувана мова, що дозволяє вчитися мислити в правильному напрямку.

Java підтримує безліч бібліотек – будівельних блоків будь-якої корпоративної системи. Бібліотеки допомагають розробникам створювати будь-які функції, які можуть знадобитися компанії. Можливості інтеграції Java вражають: більшість хостинг-провайдерів підтримують Java. Більш того, Java дешева в обслуговуванні: працювати з Java можна з будь-якого комп'ютера, незалежно від конкретної апаратної інфраструктури [10].

Середовище розробки – IntelliJ IDEA. Перевагою є безкоштовність Community версії середовища. У цьому середовищі розробки наявна система

перевірки коректності коду та система контролю за виконанням завдань. Серед переваг також є можливість інтеграції коду з GitHub.

Підтримує засіб складання проектів Maven. «Apache Maven» — це засіб автоматизації певних функцій, потрібних для роботи з програмним кодом. Він використовується для Java проектів та для управління, компіляціїї та пакетування (build) програм.

Maven використовує конструкцію Project Object Model (POM). Виконання певних, предвизначених задач – таких, як компіляція коду, інсталяція та пакетування відбувається шляхом досягнення заздалегідь визначених targets.

Ключовою особливістю Maven є його мережева готовність (network-ready).

2.2 База даних

Для серверної частини проекту було обрано MySQL, а для клієнтської – SQLite3. Вважається, що Oracle більше підходить для Java, але в даному випадку є кілька переваг MySQL, які дуже важливі. А саме:

- завдяки внутрішньому механізму багатопотоковості, швидкість цієї бази даних дуже висока;
- ця програма для некомерційних цілей розповсюджується безкоштовно;
- завдяки відкритості коду можна самостійно додати необхідні функції до пакету, розширивши його функціонал за необхідності;
- СКБД MySQL стабільна і її складно вивести з ладу;
- у даний час існують версії цього програмного забезпечення для більшості широко поширених комп'ютерних платформ. Можна обирати, з чим працювати, наприклад, Linux або Windows. Навіть у випадку зміни однієї ОС на іншу, ви не втратите свої дані і не знадобляться додаткові інструменти для їх переміщення [11].

Щодо СКБД SQLite3, то вона буде використовуватися для клієнтської частини додатку та буде створюватись та використовуватись безпосередньо у користувача. Це будуть певні дані, які потрібно зберігати, але явно не на сервері. Тому для таких функцій цієї СКБД буде досить.

Відмінною характеристикою SQLite є те, що рушій цієї СКБД представляє собою не окремий процес, з яким взаємодіє програма або додаток, а надає безпосередньо бібліотеку, з якою програма (додаток) компілюється і рушій SQLite стає частиною програми, входить до її складу. Таким чином, як протокол обміну використовуються виклики API функцій бібліотеки SQLite.

За допомогою такого підходу зменшуються час відгуку і це значно спрощує програму. СКБД зберігає базу даних (включаючи таблиці, індекси і дані) в єдиному файлі на тій машині, на якому виконується програма [12]. У даному випадку буде створюватися файл на комп'ютері користувача, який буде використовувати програму.

SQLite3 використовують у вбудованих додатки – якщо важлива можливість легкого перенесення програми, але не важлива масштабованість. А також при необхідності безпосередньо звертатися до диску можна виграти при переході на цю СУБД в функціоналі і простоті використання SQL мови [9].

JDBC (Java DataBase Connectivity) − незалежний від платформи галузевий стандарт взаємодії Java-додатків з різними SUBD, реалізований у вигляді пакету java.sql, включеного в Java SE [13]. Саме цей драйвер буде забезпечувати взаємодію нашого Java-додатку з вищенаведеними базами даних. Головною причиною використання JDBC є те, що він вже включений в Java SE та його легко підключити до проекту.

Варто відмітити також інші переваги JDBC, а саме:

- легкість розробки: розробник може не знати специфіки бази даних;
- код практично не змінюється, якщо компанія переходить на іншу базу даних (кількість змін залежить виключно від відмінностей між діалектами SQL);
- не потрібно встановлювати громіздку клієнтську програму;
- до будь-якої бази можна під'єднатися через легко описуваний URL [13].

2.3 Алгоритми шифрування

Месенжер має максимально забезпечити безпеку від взлому та викрадення повідомлень користувачів. Для цього буде використано такі технології як MD5, RSA та AES.

MD5 (Message Digest 5) – 128-бітний алгоритм хешування, призначений для створення «відбитків» або «дайджестів» повідомлень довільної довжини [14]. Цей спосіб шифрування є найпоширенішим способом захистити інформацію в сфері прикладних досліджень, а також в області розробки веб-додатків. Хеш необхідно убезпечити від всіляких хакерських атак. Одним із дієвих способів захисту є «сіль». Необхідно додати до пароля зайві випадкові символи.

Далі термін «слово» використовується для 32-бітових значень, а «байт» – для восьмибітових. Послідовності бітів можуть інтерпретуватися природним шляхом як послідовність байтів, в якій кожна група з 8 послідовних бітів інтерпретується як байт, де старший (найбільш значимий) біт розташовується першим. Подібно до цього послідовність байтів може інтерпретуватися як послідовність 32-бітових слів, в якій кожна група з 4 послідовних байтів інтерпретується як слово, де молодший (найменш значимий) вказується першим.

Символ «+» позначає складання слів (тобто, складання з використанням модуля 2^{32}). Запис $\boldsymbol{X} <<< \boldsymbol{s}$ означає 32-бітове слово, отримане циклічним зрушенням *X* вліво на *s* позицій. $\boldsymbol{not}$ ($\boldsymbol{X}$) означає побітове доповнення *X*, а $\boldsymbol{X}\ \boldsymbol{v}\ \boldsymbol{Y}$ – побітову операцію $\boldsymbol{X}\ \boldsymbol{OR}\ \boldsymbol{Y}$ (*X* АБО *Y*). Запис $\boldsymbol{X}\ \boldsymbol{xor}\ \boldsymbol{Y}$ означає побітову операцію $\boldsymbol{X}\ \boldsymbol{XOR}\ \boldsymbol{Y}$ (виключаюче АБО), а $\boldsymbol{XY}$ побітову операцію $\boldsymbol{X}\ \boldsymbol{AND}\ \boldsymbol{Y}$ (*X* І *Y*).

Опис алгоритму MD5. Почнемо з припущення про наявність на вході повідомлення розміром b бітів, для якого потрібно створити цифровий підпис. Тут b означає невід'ємне ціле число, b може приймати нульове значення і не повинно бути кратним 8. Це значення може бути необмежено великим. Біти вихідного повідомлення будемо представляти наступним чином (2.1):

$$m[0]\, m[1] \ldots m[b-1] \tag{2.1}$$

Для створення цифрового підпису виконується процес з 5 описаних нижче етапів.

Етап 1 Додавання бітів заповнення.

Повідомлення доповнюється до розміру (в бітах), конгруентного 448 по модулю 512. Тобто, розмір повідомлення встановлюється таким, щоб після додавання 64 бітів розмір був кратний 512. Заповнення проводиться у всіх випадках, навіть якщо розмір вихідного повідомлення конгруентний 448 по модулю 512.

Заповнення відбувається наступним чином – спочатку в кінці повідомлення додається один біт, що має значення 1, а потім додаються біти, що мають значення 0 до розміру, конгруентного 448 по модулю 512. У будь-якому випадку число додаються бітів не може бути менше 1 і більше 512.

Етап 2 Додавання розміру повідомлення.

64-бітове представлення b (розмір вихідного повідомлення в бітах) додається в кінець результату попереднього етапу. У рідкісних випадках, коли b більше 264, використовуються тільки молодші 64 біта b. Біти додаються в кінці, як два 32-бітових слова, в яких молодший байт розміщується на початку, як було домовлено вище. Після цього розмір отриманого повідомлення (вихідне повідомлення + заповнення + розмір) буде кратний 512 бітам. Отже, результуюче повідомлення буде містити ціле число блоків по 16 слів (32 біта в кожному слові). Нехай $M\ [0 \ldots N-1]$ позначає слова отриманого в результаті повідомлення (N кратне 16).

Етап 3 Ініціалізація буфера MD.

Для розрахунку цифрового підпису використовується буфер на 4 слова (*A, B, C, D*). Кожне з слів *A, B, C, D* являє собою 32-бітовий регістр. Ці регістри ініціалізуються наведеними нижче значеннями (шістанадцяткове представлення, спочатку молодший байт):

слово A: 01 23 45 67

слово B: 89 ab cd ef

слово C: fe dc ba 98

слово D: 76 54 32 10

Етап 4 Обробка повідомлення блоками по 16 слів.

Спочатку визначимо чотири додаткових функції 2.2 – 2.5, кожна з яких бере три 32-бітових слова в якості аргументів і повертає одне 32-бітове слово:

$$F\ (X, Y, Z) =\ XY\ v\ not\ (X)Z \tag{2.2}$$

$$G\ (X, Y, Z) =\ XZ\ v\ Y\ not\ (Z) \tag{2.3}$$

$$H\ (X, Y, Z) =\ X\ xor\ Y\ xor\ Z \tag{2.4}$$

$$I\ (X, Y, Z) =\ Y\ xor\ \big(X\ v\ not\ (Z)\big) \tag{2.5}$$

Для кожного біта функція *F* працює як умова – якщо *X*, то *Y*, інакше *Z*. Функцію *F* можна визначити з використанням складання (+) замість операції АБО (*v*), оскільки *XY* і *not (X) Z* ніколи не будуть давати 1 в одній бітової позиції. За умови, коли біти *X*, *Y* і *Z* незалежні і не утворені зміщенням, кожен біт функції *F (X, Y, Z)* буде незалежним і не буде утворений зміщенням.

Функції *G*, *H*, *I* схожі на функцію *F* – фактично вони діють «паралельно по бітам» для створення результатів за значеннями *X*, *Y*, *Z* так, щоб при незалежних і не утворених зміщенням бітах *X*, *Y* і *Z* кожен біт *G* (*X*, *Y*, *Z*), *H* (*X*, *Y*, *Z*) і *I* (*X*, *Y*, *Z*) також був незалежним і не виходив шляхом зміщення. Відзначимо, що функція *H* є побітову операцію «виключає АБО» (*xor*) або функцію «парності» переданих їй аргументів.

На цьому етапі використовується 64-елементна таблиця $T[1 \dots 64]$, побудована з використанням синусоїдальної функції. Нехай *T[i]* позначає і-й елемент таблиці, який дорівнює цілій частині добутку $4294967296 \times abs\ (sin\ (i))$, де *i* задано в радіанах.

Етап 5 Виведення.

Сигнатура повідомлення виводиться як *A*, *B*, *C*, *D* (тобто, починаємо висновок з молодшого байта *A* і закінчуємо старшим байтом *D*) [15]. Алгоритм цифрових підписів MD5 простий в реалізації і створює «відбитки» або цифрові підписи для повідомлень довільної довжини. Передбачається, що для створення двох повідомлень з однаковими сигнатурами буде потрібно близько 2^{64} операцій,

а для підбору повідомлення за наявною сигнатуре – близько 2^{128} операцій [15]. Алгоритм MD5 був ретельно досліджений на предмет пошуку слабких місць. Однак алгоритм є досить новим і потрібно додатковий аналіз, як і для всіх новинок цього типу.

“RSA (абревіатура від прізвищ Rivest, Shamir і Adleman) – криптографічний алгоритм з відкритим ключем, який базується на обчислювальної складності задачі факторизації великих цілих чисел” [16]. На сьогодні RSA – відома криптосистема, яка підтримує більшість електронних комерційних комунікацій.

Якщо коротко, то шифрування RSA працює за нескладним алгоритмом (рис. 2.1). Перш за все йде запит відкритого ключа з сервера. Далі дані користувача зашифровуються за допомогою цього відкритого ключа. Потім відбувається розшифрування даних за допомогою секретного ключа на сервері. Варто сказати, що знаючи відкритий ключ повідомлення не можна розшифрувати.

Рисунок 2.1 – Принцип роботи RSA шифрування

Advanced Encryption Standard (AES) – симетричний алгоритм блочного шифрування. Найвикористовуванішим на даний момент є змішаний алгоритм шифрування. У ньому у першу чергу шифрується сеансовий ключ, а потім з його допомогою користувачі шифрують свої повідомлення симетричними системами. Як правило, після завершення сеансу цей сеансовий ключ знищується. Це реалізується за допомогою RSA та AES, які і будуть використані в моїй роботі.

Нижче наведений принцип змішанного алгоритму шифрування (рис. 2.2):

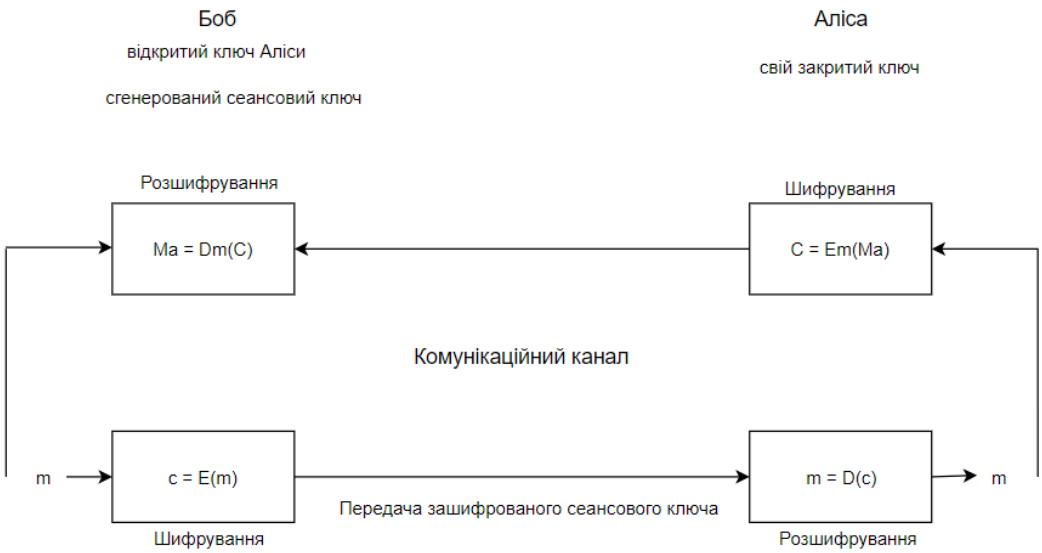

Рисунок 2.2 – Робота алгоритму шифрування сеансового ключа

Алгоритм Боба:

1. Взяти відкритий ключ (e, m) Аліси;

2. Створити випадковий сеансовий ключ m;

3. Зашифрувати сеансовий ключ з використанням відкритого ключа Аліси за формулою 2.6:

$$c = E(m) = m^e \ mod \ n; \quad (2.6)$$

4. Розшифрувати повідомлення C за допомогою сеансового ключа симетричним алгоритмом (формула 2.5):

$$M_A = D_m(C). \ (2.5)$$

Алгоритм Аліси:

1) Прийняти зашифрований сеансовий ключ Боба c;

2) Взяти свій закритий ключ (d, n);

3) Застосувати закритий ключ для розшифрування сеансового ключа (формула 2.6):

$$m = D(c) = c^d \ mod\ n; \tag{2.6}$$

4) Зашифрувати повідомлення M_A за допомогою сеансового ключа симетричним алгоритмом (формула 2.7):

$$C = E_m(M_A) \tag{2.7}$$

У разі, коли даний сеансовий ключ більше, ніж модуль n, його розбивають на блоки потрібної довжини (в разі необхідності доповнюють нулями) і шифрують кожен такий блок [16].

2.4 Брокер повідомлень

“RabbitMQ – платформа, що реалізує систему обміну повідомленнями між компонентами програмної системи на основі стандарту AMQP (Advanced Message Queuing Protocol)” [17].

Це додаток для роботи з чергами повідомлень (message-queueing), ще його називають меседж брокер (message broker) або менеджер черг (queue manager). Простими словами – це програмне забезпечення, в якому можуть бути визначені черги, до якого можуть підключаться різні додатки і передавати/отримувати повідомлення [18]. Сервер RabbitMQ по суті є менеджером черг, який має такі переваги:

- в разі некоректного завершення роботи сервера, дані в черзі не втрачаються. І при наступному запуску обробка триває з того місця, де був обрив;

- розподілити завдання на кілька черг, тобто створити розпаралелювання на рівні повідомлень;
- якщо результат обробки не задовольняє, завдання можна послати в чергу повторно;
- можливість синхронізувати роботу клієнта і сервера, свого роду реалізація RPC;
- кількість збережених в черзі повідомлень необмежена [19].

Задача розробки полягатиме ще й в тому, щоб максимально забезпечити безпеку від взлому та викрадення повідомлень користувачів. Для цього буде використано такі технології як MD5, RSA та AES.

Повідомлення може включати в себе будь-яку інформацію. Для прикладу у нас є інформація про процес, який має розпочати іншу програму (або ж інший сервер), або це може бути простим текстовим повідомленням. Менеджер черг – додаток, який зберігає повідомлення до тих пір поки інший додаток (сервер), якому адресовано повідомлення, не підключитися і не забере (отримає) повідомлення з черги. Потім додаток-одержувач обробить повідомлення потрібним йому чином [18]. Нижче представлена вищеописана схема роботи (рис. 2.3):

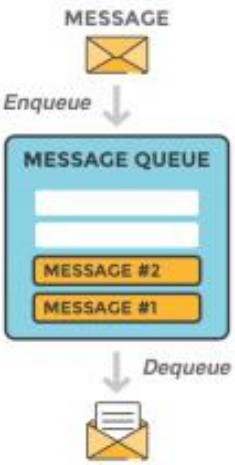

Рисунок 2.3 – Схема того, як повідомлення ставиться в чергу та виходить з неї в RabbitMQ

Базова архітектура черги повідомлень досить проста, клієнтська програма (Producer) створює повідомлення і доставляє його в меседж брокер. Інша програма (Consumer) одержувач підключається до черги і підписується на повідомлення, які має обробити [18]. Програма може бути як постачальником повідомлень, так і одержувачем або ж і тим і іншим одночасно. Схематичне представлення ролі RabbitMQ в процесі обміну повідомлень між Producer та Consumer на рисунку 2.4.

Рисунок 2.4 – Схематичне зображення ролі RabbitMQ в процесі обміну повідомлень

Черги повідомлень дозволяють серверам отримувати/відправляти запити швидше, замість того, щоб очікувати виконання ресурсоємних процедур. Черга повідомлень також добре застосовується до задачі відправки повідомлень кільком одержувачам для зниження споживання пам'яті або збалансування навантаження.

Повертаючись до вищеприведеного прикладу одержувач може взяти повідомлення з черги і почати обробляти, в той час, коли постачальник додає нові повідомлення в чергу. Одержувачем може бути зовсім інший сервер, відмінний від сервера постачальника, або ж вони можуть перебувати на одному і тому ж. Запит може бути створений на одній мові програмування і оброблений іншим – вони "спілкуються" тільки через повідомлення, які посилають один одному [18]. Виходячи з цього, додатки будуть мати низький зв'язок між відправником і отримувачем, більш незалежними.

Короткий опис принципу роботи брокеру повідомлень на прикладі:

1. Користувач посилає запит на створення PDF.

2. Додаток (Producer) посилає повідомлення в RabbitMQ, включаючи в запит інформацію, наприклад, ім'я та електронну пошту.

3. Оброблювач приймає повідомлення від програми постачальника і направляє його в потрібну чергу повідомлень.

4. Воркер обробки PDF (Consumer) отримує завдання – почати генерацію PDF [18].

На рисунку 2.5 зображено схему роботи RabbitMQ на прикладі:

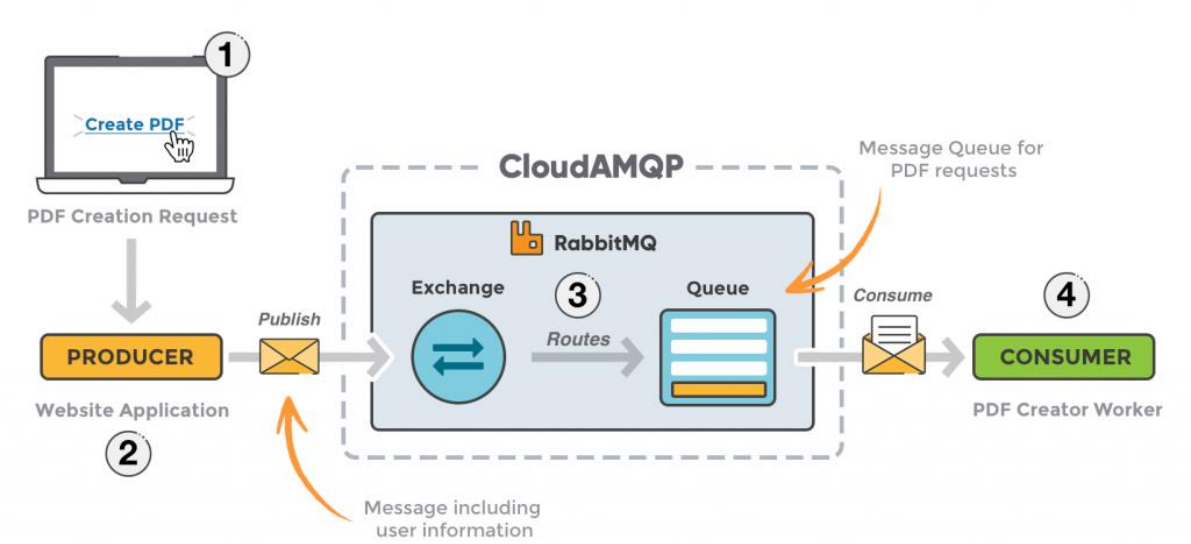

Рисунок 2.5 – Принцип роботи RabbitMQ

Оброблювач повідомлень. Повідомлення публікуються в чергу не на пряму, замість цього, постачальник шле повідомлення оброблювачу, який відповідає за перенаправлення повідомлення в потрібну чергу за допомогою біндінгу ключів роутінгу.

Схема потоку повідомлень у RabbitMQ:

1. Постачальник публікує повідомлення в обробник. Коли створюємо обробник ми визначаємо його тип.

2. Оброблювач отримує повідомлення і відповідає за його перенаправлення. Оброблювач бере різні атрибути, такі як, ключ роутінгу, залежність на тип обміну та інші.

3. Створюється зв'язок між оброблювачем і чергою.

4. Повідомлення залишається в черзі до тих пір, поки воно не буде оброблено одержувачем.

5. Одержувач обробляє повідомлення [18].

Схема потоку повідомлень зображена на рис.2.6.

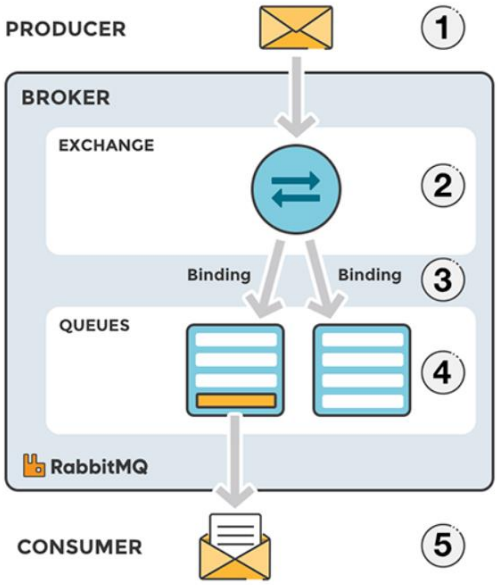

Рисунок 2.6 – Потік повідомлень в RabbitMQ

Вищенаведені технології та бази даних допоможуть реалізувати додаток для миттєвого обміну повідомленнями.

3. ПРОЕКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

3.1 Розробка алгоритмів функціонування системи

Нижче приведені схеми (рис. 3.1 – 3.3), що ілюструють взаємодію користувача із системою, логічні послідовності дій основної функціональної складової нашої програми.

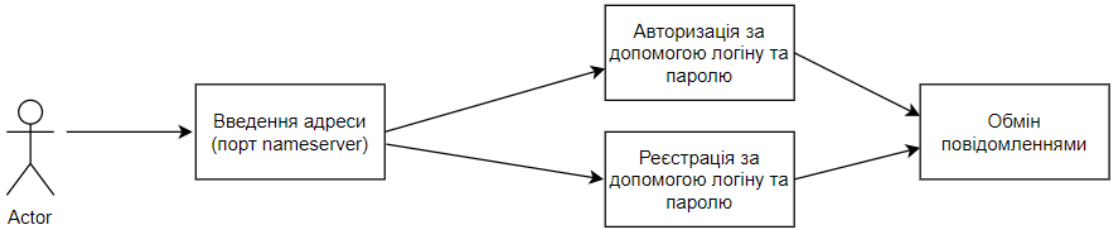

Рисунок 3.1 – Процес входу користувача до свого акаунту в програмі

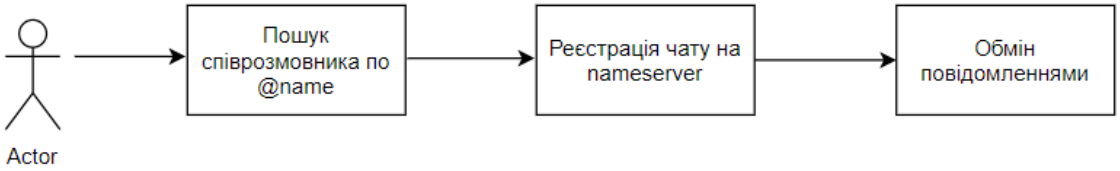

Рисунок 3.2 – Початок нового чату з новим для акаунту користувачем

Рисунок 3.3 – Продовження спілкування з певним користувачем

Нижче приведена схема (рис. 3.4) обміну повідомленнями між користувачами. Повідомлення відправляється nameserver, звідки відправляється на dataserverX, де зберігається зашифрованим. Потім nameserver відправляє

запит на всі dataserver для пошуку другого учасника чату. Коли певний dataserverW відповідає на запит, то йому відправляється повідомлення.

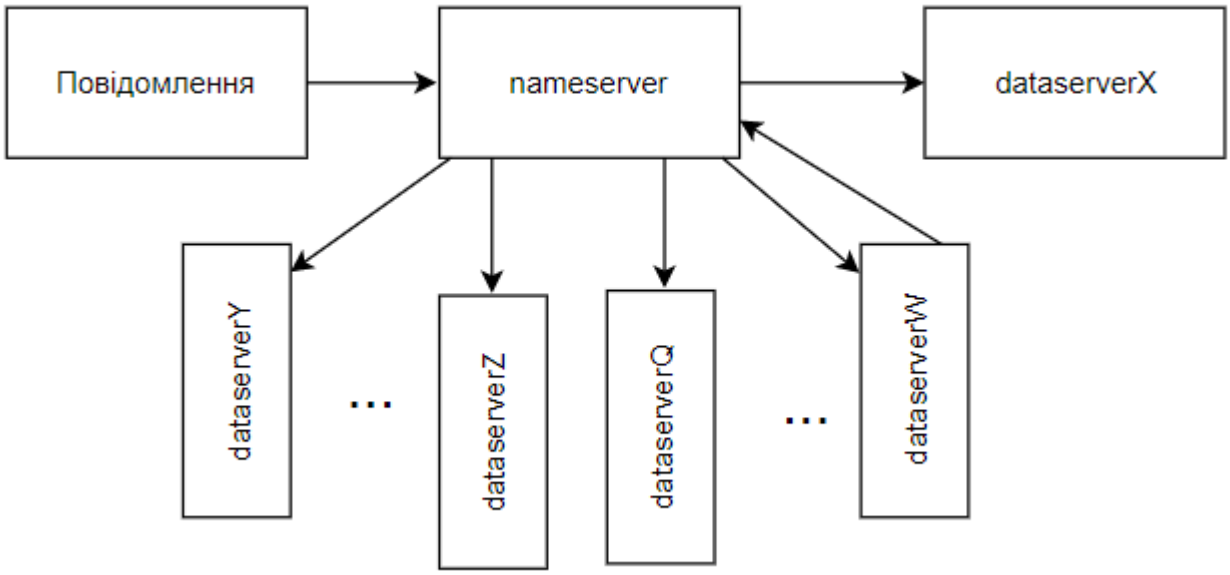

Рисунок 3.4 – Пошук користувача, якому відправити повідомлення

3.2 Проектування алгоритмічного та програмного забезпечення

Nameserver – сервер імен призначений для реєстрації імен користувачів і серверів даних в мережі. Є центральним вузловим пунктом, що відповідає за авторизацію і пошук по імені. По суті є віддаленим менеджером бази даних (JDBC + MySQL).

Його схема роботи: він слухає підключення на 3486 (або іншому вказаному на початку) порту. Відповідає на повідомлення по протоколу. У результаті порушення протоколу відповідає повідомленням помилки (ERROR). Не підтримує тривалі підключення (тільки формату запит1 – відповідь1). Може відправляти запити типу PING серверів даних.

Transfer Protocol – протокол спілкування компонентів програми.

Структура повідомлення:

--header--

type 1 byte

session 32 bytes

--body--

depends on message type (JSON)

--end--

hash (header + body) (md5 sum) 32 bytes

Опис роботи протоколу:

Повідомлення приходить стисненим алгоритмом deflate. Відповідно потрібно декомпресувати дані.

За номером заголовка обчислюється тип повідомлення (1 байт). Далі слідують 32 байти ідентифікаційного номера сеансу (якщо він не встановлений – значення 0). Якщо сеанс не встановлений, значить тіло повідомлення не зашифроване. Якщо встановлений – потрібно знайти ключ шифрування в базі даних сеансу, щоб розшифрувати тіло повідомлення. Якщо ключ не знайдений, слід відправити помилку про некоректність номеру сеансу (навіть якщо формат дотриманий – нам не потрібно говорити, що такий не існує в базі даних).

Тіло повідомлення є JSON, який містить інформацію з певним набором полів, які визначаються типом повідомлення. Тіло перетворюється в Java об'єкт для обробки.

Після обробки формується у відповідь повідомлення зворотним чином. Java об'єкт повідомлення перетворюється в JSON. Він зашифрований (якщо був встановлений сеанс) і додається в повідомлення. Тема формується з типу повідомлення та ідентифікаційного номеру сеансу (якщо він був). Одержаний набір байт стискається алгоритмом deflate і відправляється.

Типи повідомлень і опис форматів:

ERROR помилка. Номер типу: 0 Може шифруватися. Формат тіла повідомлення:

{"Code": 100, // код помилки

"Text": "" // текст помилки}

PING перевірка кінцевого адресата. Номер типу: 1 Не шифрується. Формат тіла повідомлення:

{ "Waiting_time": 600, // час очікування відповідного пакету в мілісекундах
"Sent_time": 0,
"Payload": '00000000' // корисне навантаження, 1 байт. Повинно бути у відповідному пакеті}

PONG відповідь на пакет PING. Номер типу: 2 Не шифрується. Формат тіла повідомлення:

{ "Received_time": 0,
"Payload": '00000000' // повинен містити таке ж корисне навантаження, як і в PING повідомленні}

GET_PUBKEY запит на отримання публічної ключа. Номер типу 3 Не шифрується. Формат тіла повідомлення:{}

PUBKEY відповідь на запит на отримання публічної ключа. Номер типу: 4 Не шифрується. Формат тіла повідомлення:

{ "Algorithm": "RSA", // алгоритм шифрування
"Size": 2048, // розмір ключа
"Key": "" // публічний ключ}

SET_SESSIONKEY встановлення сеансового ключа для шифрування. Номер типу: 5 Шифрується публічним ключем (RSA). Формат тіла повідомлення:

{ "Algorithm": "AES", // назва алгоритму
"Size": 256, // розмір ключа алгоритму
"Key": "" // сеансовий ключ для шифрування трафіку}

SESSION_INFO інформація про сеанси. Номер типу: 6 Шифрується сеансовим ключем (AES). Формат тіла повідомлення:

{ "Id": "", // ідентифікаційний номер сеансу
"Status": "OPEN | CLOSED" // статус сеансу}

CLOSE_SESSION закриття сеансу. Номер типу: 7 Шифрується сеансовим ключем (AES). Формат тіла повідомлення

{ "Id": "" // ідентифікаційний номер сеансу, який необхідно завершити}

GET_USER_INFO отримати інформацію про користувача за його ідентифікаційним номером або іменем. Номер типу: 8 Шифрується сеансовим ключем (AES). Формат тіла повідомлення:

{ "Id": "" // ідентифікаційний номер користувача (опціонально)

"Name": "" // ім'я користувача (опціонально)}

USER_INFO інформація про користувача. Номер типу: 9 Шифрується сеансовим ключем (AES). Формат тіла повідомлення:

{ "Id": "", // ідентифікаційний номер користувача

"Name": "" // ім'я користувача}

REG_USER запит на реєстрацію користувача. Номер типу: A Шифрується сеансовим ключем (AES). Формат тіла повідомлення:

{ "Session_id": "", // ідентифікаційний номер сеансу

"Name": "", // ім'я користувача

"Password": "" // пароль користувача

"Pubkey": "" // публічний ключ RSA}

В результаті вдалої реєстрації – USER_INFO, інакше – ERROR.

AUTH_USER запит на авторизацію в мережі. Номер типу: B Шифрується сеансовим ключем (AES). Формат тіла повідомлення:

{ "Session_id": "", // ідентифікаційний номер сеансу

"Name": "", // ім'я користувача

"Password": "" // пароль користувача}

В результаті вдалої авторизації – USER_INFO, інакше – ERROR. На сервері імен встановлюється значення активного сеансу користувача.

3.3 Архітектура та опис модулів

Реалізація бази даних з поясненнями зв’язків приведена нижче.

NameServer таблиці:

nameserver (клас NameServerDao):

У даній таблиці 3.1 міститься інформація про зареєстрований сервер імен (nameserver) у локальній мережі. Доступний клієнтам для того, щоб дізнаватися інформацію про сервер імен у першому вікні програми, отримання відкритого ключа для шифрування з'єднання, а також складається з приватного ключа, який є частиною раніше (при першому завантаженні) згенерованого ключа пари.

Таблиця 3.1 – Nameserver таблиця

id	address	pubkey	privkey
int	NS address:port url	text	text

dataservers (клас DataServerDao):

У даній таблиці 3.2 міститься інформація про зареєстровані сервера даних (dataservers) у локальній мережі.

Таблиця 3.2 – Dataservers таблиця

id	name	address	pubkey
int	@randomUUID	DS address:port url	text

sessions (клас SessionDao):

У таблиці 3.3 міститься інформація про пряме з'єднання користувача з NS, що використовується для шифрування з'єднання з AES. По-перше, через зашифрований канал RSA створюється новий сеанс, потім з'єднує його з конкретним користувачем. Ідентифікатор сесії, що використовується для доступу до ключа шифрування (для AES). Статус (відкритий / закритий) повідомляє про поточний стан сеансу. Користувач не може отримати доступ до приватного ключа та авторизуватися ним, якщо сеанс закритий. Коли створюється сессія, вона автоматично рахується відкритою.

users (клас UserDao):

У таблиці 3.4 міститься інформація про зареєстрованого користувача, в нього може і не бути електронної пошти. Пароль – це фактично контрольна сума

md5 із пароля користувачів; публічний ключ зберігається для передсесійного з'єднання RSA.

Таблиця 3.3 – Sessions таблиця

id	user_id	status	key
randomUUID	int	Boolean (is open)	text

Таблиця 3.4 – Users таблиця

id	name	password	email	pubkey
Int	@text	Md5(text)	null	text

chats (клас ChatDao):

Тут (таблиця 3.5) міститься інформація про зареєстровані чати між двома користувачами (ініціатором та консолідатором).

initiator_id (foreign key -> user_id) = user id того, хто створив (ініціював) чат; consolidator_id (foreign key -> user_id) = user id того, хто підтвердив створення такого чату; initiator_secret_key = RSA зашифрований ключ для з'єднання AES (сеансового), згенерований консолідатором для ініціатора (відкритим ключем), коли консолідатор приймає чат від ініціатора; consolidator_secret_key = те саме, що і попереднє, але ключ згенерований ініціатором під час створення чату; last_message_id = id останнього повідомлення в чаті, звідки буде завантаження (оскільки завантаження йде від останнього повідомлення до першого, вони не можуть бути завантажені всі разом за один запит. Таким чином вони завантажуються, коли користувач прокручує чат вгору).

Таблиця 3.5 – Chats таблиця.

id	Initiator_id	Consolidator_id	Initiator_secret_key	Consolidator_secret_key	Last_message_id
int	int	int	text	text	int

DataServer таблиці:

dataserver:

ВУ поданій нижче таблиці 3.6 міститься інформація про локально зареєстрований сервер даних (dataserver), у випадку локально зареєстрованого сервера імен.

Таблиця 3.6 – Dataserver таблиця

id	address	pubkey	privkey
int	DS address:port URL	text	Text

messages:

У таблиці 3.7 зберігаються збережені зашифровані повідомлення, де id – це номер повідомлення в чаті; chat_id – це чат, який складається з цього повідомлення та тексту; text – текст, що зашифрований AES (текст повідомлення).

Таблиця 3.7 – Messages таблиця

id (NOT PRIMARY KEY)	chat_id	text
int	int	AES(text)

Client таблиці:

nameservers:

Дана таблиця (табл. 3.8) містить інформацію про nameserver, його ідентифікаційний номер (id), адресу та порт. Всі три поля повинні бути заповнені (NOT NULL).

Таблиця 3.8 – Nameservers таблиця

nameserver_id	address	port
integer	text	text

users:

Дана таблиця (табл. 3.9) має інформацію про користувача, його id та ім'я, які не можуть бути нульовими, тобто пустими. Окрім того, є foreign key з таблиці nameservers, щоб зв’язати користувача з сервером імен.

Таблиця 3.9 – Users таблиця

user_id	name	nameserver_id
integer	text	Foreign key → nameservers

sessions:

Подана нижче таблиця 3.10 ілюструє, як представлений сеанс у базі даних. Ми бачимо, що сюди входить номер сеансу та ключ, на яких накладене обмеження NOT NULL. Також містяться поля nameserver_id та user_id – foreign keys з таблиць nameservers та users відповідно.

Таблиця 3.10 – Sessions таблиця

session_id	key	nameserver_id	user_id
text	text	Foreign key → nameservers	Foreign key → users

keys:

Таблиця 3.11, що представлена нижче, містить інформацію про ключ, а саме поля: ідентифікаційний номер пари ключів (публічного та приватного), публічний ключ та приватний ключ, які повинні бути заповнені. Також в цій таблиці є foreign keys – nameserver_id, user_id.

Таблиця 3.11 – Keys таблиця

key_pair_id	pubkey	privkey	nameserver_id	user_id
integer	text	text	Fkey → nameservers	Fkey → users

chats:

Таблиця нижче (табл. 3.12) містить інформацію про id чату, ініціатор, консолідатор, ключ, дані про те, чи підтверджено чат консолідатором та останнє проглянуте повідомлення. Також містяться поля nameserver_id та user_id – foreign keys з таблиць nameservers та users відповідно.

Таблиця 3.12 – Chats таблиця

chat_id	initiator_id	consolidator_id	key	accepted	last_seen_message_id	nameserver_id	user_id
int	int	int	text	boolean	int	Fkey → nameservers	Fkey → users

У додатку А приведена таблиця з назвою пакетів (package) та опис їх призначення у програмній реалізації.

3.4 Дизайн інтерфейсу програмного продукту

На рисунках, що будуть далі, показано, який вигляд діалогових вікон програми. Дана програма має 4 діалогових вікна:

- вікно входу (початкове вікно);
- вікно додавання nameserver;
- вікно повідомлень та чатів;
- вікно інформації про програму.

На кожній формі є кнопки (елементи керування) та текстові надписи. На рис. 3.5 – 3.11 зображено макети даних форм з елементами керування на них розроблених у програмі SceneBuilder-10.0.0. Нижче приведені підписи щодо кожного елементу керування.

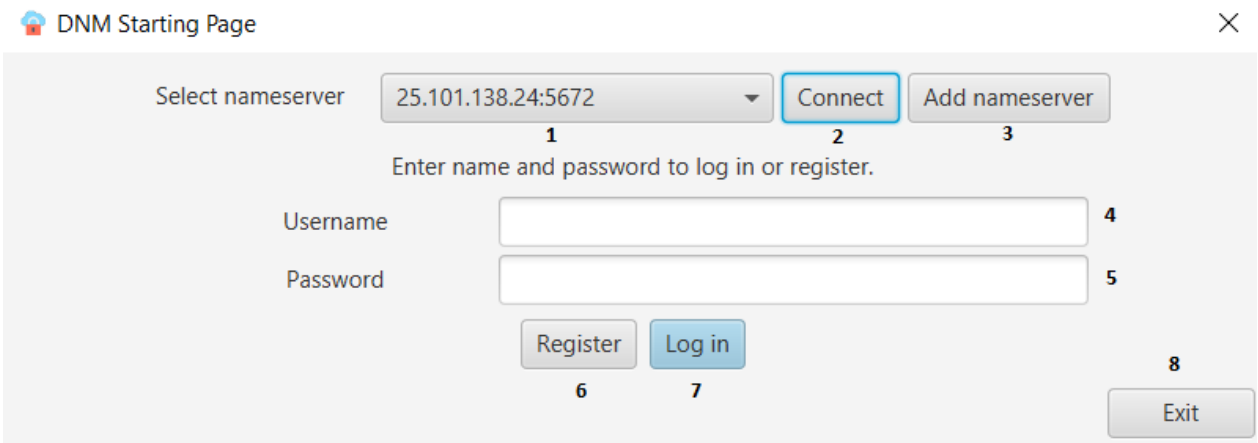

Рисунок 3.5 – Початкове вікно (вікно входу)

1 – Випадаючий список з nameservers, до який можна підключитися.

2 – Кнопка, щоб підключитися до вибраного nameserver.

3 – Кнопка для додавання власного nameserver.

4 – Поле, в яке вводиться ім’я користувача.

5 – Поле, в яке вводиться пароль користувача.

6 – Кнопка для реєстрації користувача.

7 – Кнопка для входу користувача.

8 – Кнопка для виходу з програми.

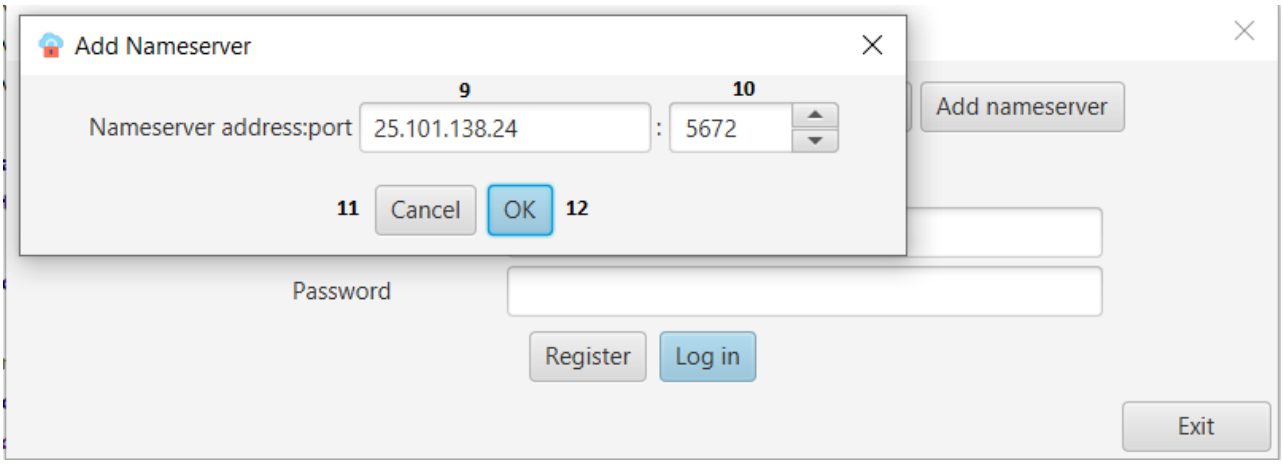

Рисунок 3.6 – Вікно додавання nameserver

9 – Поле для введення адреси серверу імен (nameserver).

10 – Поле для введення порту серверу.

11 – Кнопка відміни дій та закриття вікна.

12 – Кнопка підтвердження дій.

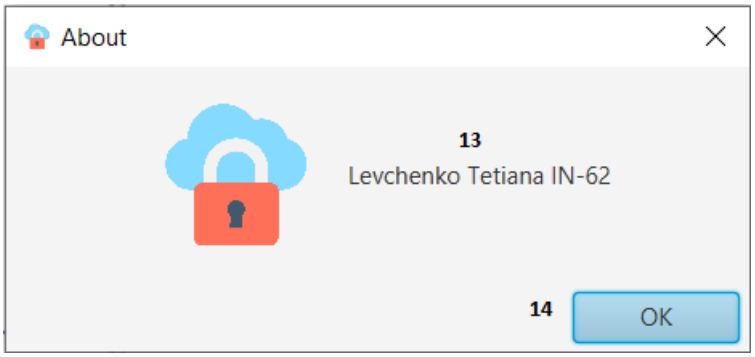

Рисунок 3.7 – Вікно інформації про програму

13 – Текст повідомлення, інформація про програму.

14 – Кнопка OK, вихід з вікна.

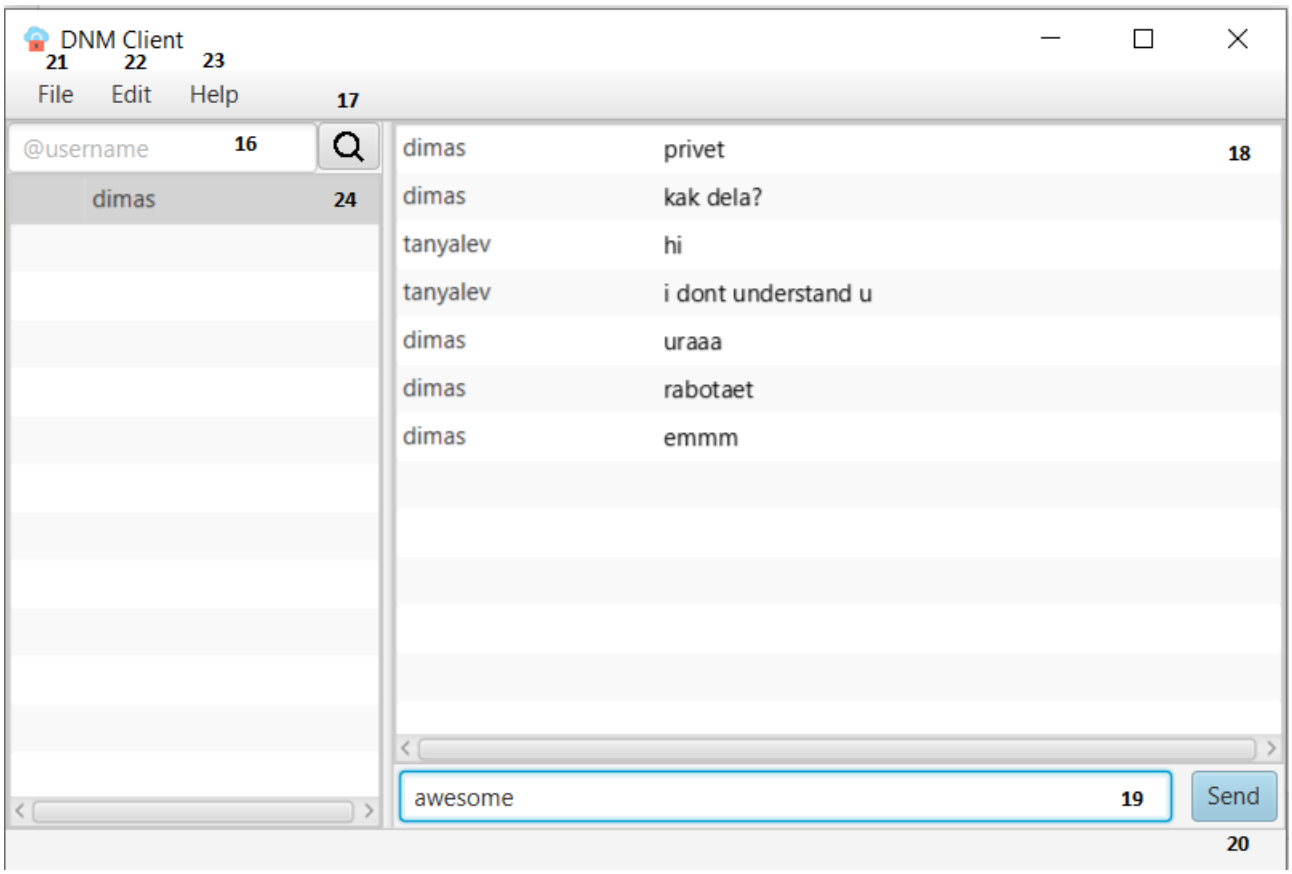

Рисунок 3.8 – Вікно повідомлень та чатів

15 – Панель зі списком чатів користувача.

16 – Поле для введення логіну користувача для пошуку.

17 – Кнопка пошуку користувача.

18 – Поле з повідомленнями обраного чату.

19 – Поле для введення нового текстового повідомлення.

20 – Кнопка для відправки введеного текстового повідомлення.

21 – Елемент панелі управління, в якому міститься вихід в програми та вихід з аккаунту.

22 – Елемент панелі управління, в якому містяться функції редагування та видалення повідомлень.

23 – Елемент панелі управління, який викликає вікно довідкової інформації про програму.

24 – Поле для відображення логіну користувача, з яким є чат.

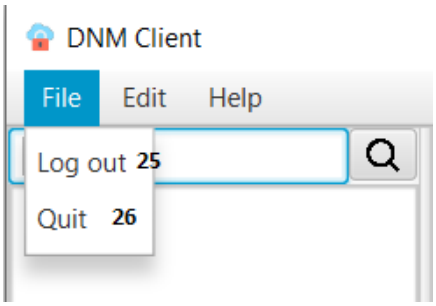

Рисунок 3.9 – Панель елементів File

25 – Функція для виходу з сеансу.

26 – Функція для виходу з програми.

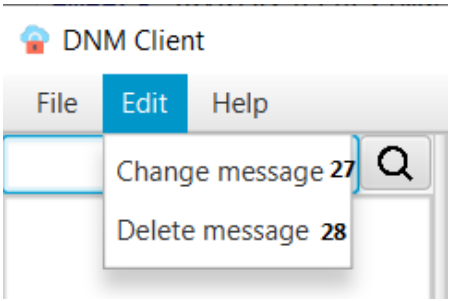

Рисунок 3.10 – Панель елементів Edit

27 – Функція для редагування повідомлення.

28 – Функція для видалення повідомлення.

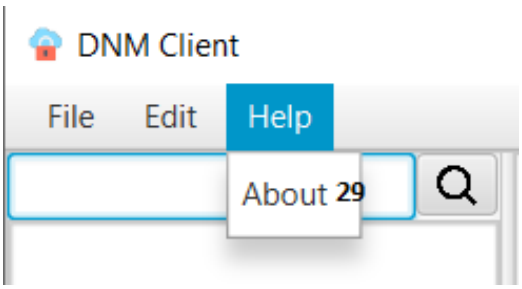

Рисунок 3.11 – Панель елементів Help.

29 – Функція для виводу інформації про програму.

Додаток було реалізовано з усіма необхідними функціями для користування.

3.5 Тестування роботи програмного продукту

Додаток було протестовано на правильність роботи основних функцій.

Тест кейс №1 «Спроба під’єднатися без серверу імен (негативна перевірка)»

1. Завантажити програму.
2. Натиснути кнопку OK.

AR=ER: Під’єднання не сталось. Відображається вікно помилки (рис. 3.12).

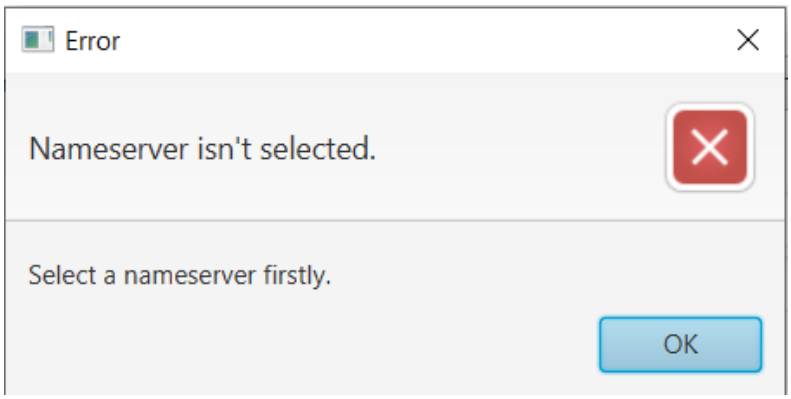

Рисунок 3.12 – Помилка під'єднання.

Тест кейс №2 «Додавання nameserver»

Пререквізити: працюючий сервер (в даному тест кейсі взято 25.101.138.24:5672).

1. Завантажити програму.
2. Натиснути кнопку Add nameserver.
3. Ввести значення адреси серверу імен у поле:

Nameserver address = 25.101.138.24

4. Ввести значення порту серверу імен:

Port = 5672

5. Натиснути кнопку OK.

AR=ER: Успішно додано адресу серверу імен до списку (рис. 3.13, 3.14).

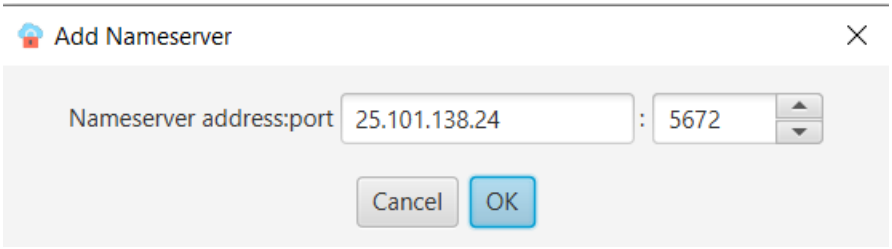

Рисунок 3.13 – Додавання nameserver.

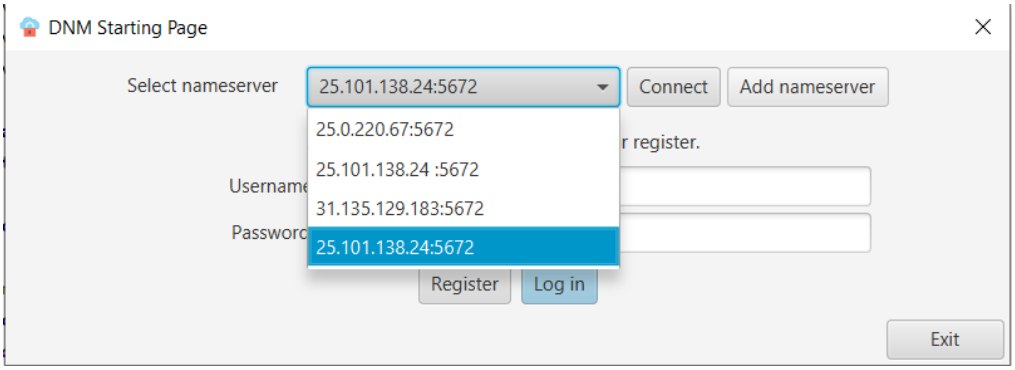

Рисунок 3.14 – Сервер успішно доданий до списка.

Тест кейс №3 «Підключення до серверу»

Пререквізити: nameserver = 25.101.138.24:5672 у базі даних.

1. Обрати сервер зі списку:

25.101.138.24:5672

2. Натиснути кнопку Connect.

AR=ER: Успішне під’єднання до сервера. Виведення повідомлення про дозвіл на заповнення логіну та паролю та активація полів для заповнення даних (рис. 3.15).

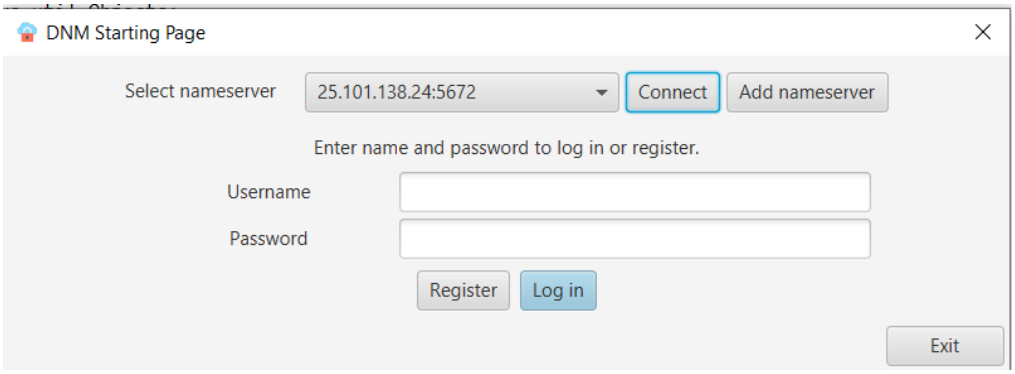

Рисунок 3.15 – Успішне під'єднання до сервера.

Тест кейс №4 «Реєстрація нового користувача»

1. Ввести в поля логін та пароль наступні значення:

Username = tanyalev

Password = 789630pf

2. Натиснути кнопку Register.

AR=ER: Реєстрацію акаунту користувача виконано успішно. Відображається повідомлення про реєстрацію користувача та можливість авторизуватися у додатку (рис.3.16).

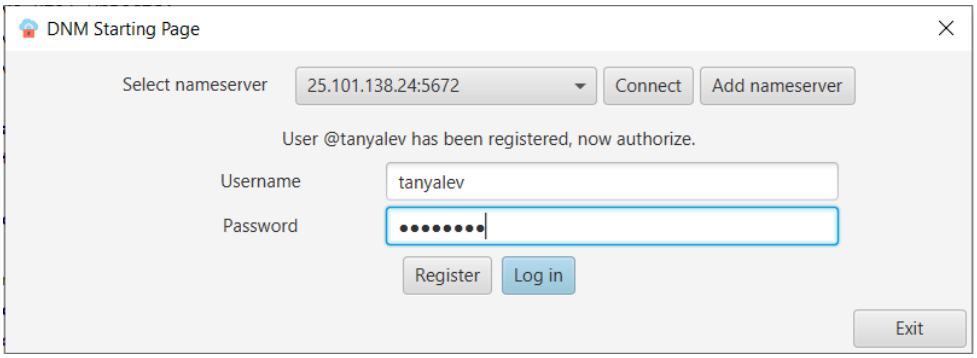

Рисунок 3.16 – Реєстрація нового користувача.

Тест кейс №5 «Авторизація користувача»

Пререквізити: Користувач з username = tanyalev та password = 789630pf у базі даних.

1. Ввести в поля логін та пароль наступні значення:

Username = tanyalev

Password = 789630pf

2. Натиснути кнопку Log in.

AR=ER: Вхід до акаунту користувача виконано успішно. Відображається вікно чатів та повідомлень (рис. 3.17).

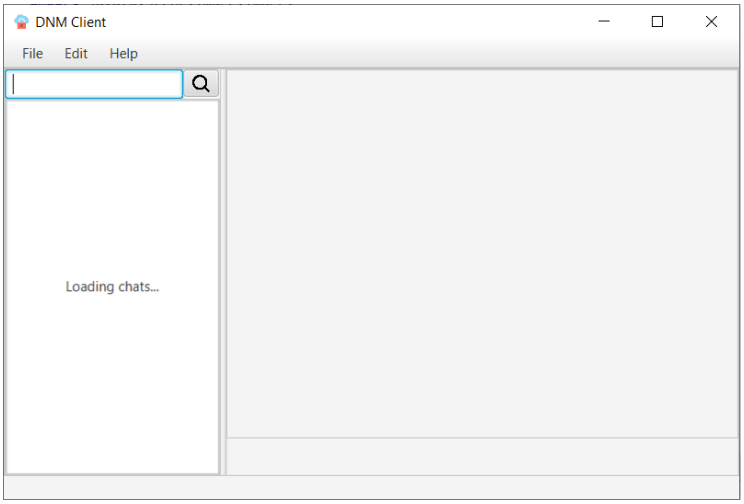

Рисунок 3.17 – Вікно чатів та повідомлень користувача.

Тест кейс №6 «Пошук користувача»

Пререквізити: Користувач з username = tanyalev та password = 789630pf у базі даних. Користувач з username = dimas.

1. Увійти в акаунт tanyalev.

2. У поле пошуку ввести значення:

dimas

AR=ER: Відображення чату з користувачем dimas (рис. 3.18).

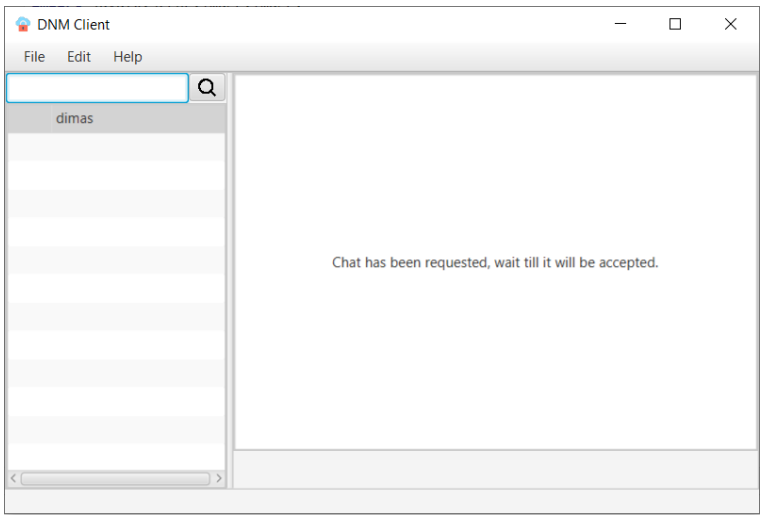

Рисунок 3.18 – Вікно чатів та повідомлень.

Тест кейс №7 «Відправка нового повідомлення»

Пререквізити: Користувач з username = tanyalev та password = 789630pf у базі даних. Користувач з username = dimas. Чат між вищенаведеними користувачами.

1. Увійти в акаунт tanyalev.
2. Написати текст повідомлення в поле для введення повідомлення (рис. 3.19).
3. Натиснути кнопку Send.

AR=ER: Успішно відправлене повідомлення (рис. 3.20).

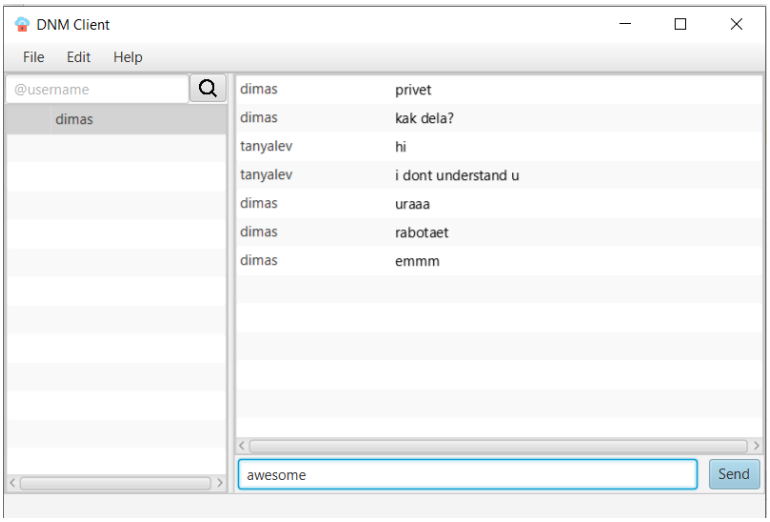

Рисунок 3.19 – Введенення тексту повідомлення.

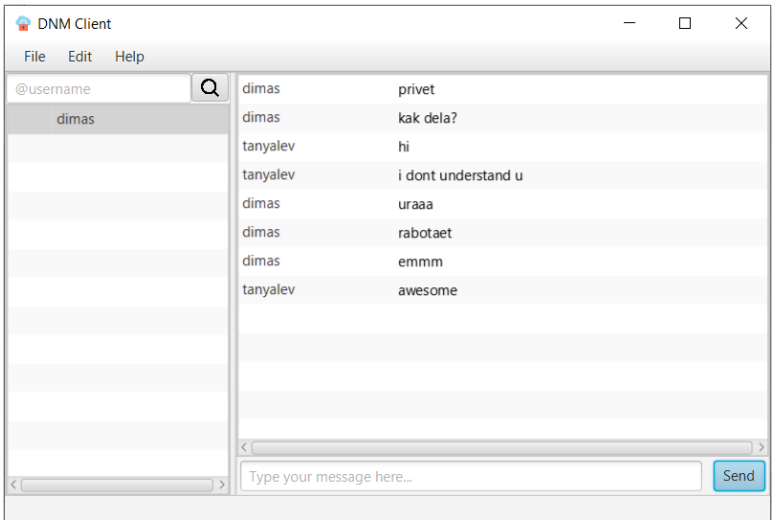

Рисунок 3.20 – Успішно відправлене повідомлення відображається в чаті.

Додаток протестовано на відповідність правильній роботі основних функцій та можливостей. Тестування успішно пройдено. Програму реалізовано у повному запланованому обсязі.

ВИСНОВКИ

У результаті виконання випускної роботи реалізовано програмний продукт для можливості обміну повідомленнями. Дану програму можна буде використовувати як звичайним користувачам, так і компаніям, яким важливий захист повідомлень. Можливість використання свого серверу імен, а не серверу додатку, повністю реалізована.

Не всі аналоги можуть забезпечити повну безпеку повідомлень. Всім вже відомо, що такі месенджери як WhatsApp, Messanger Facebook, Vkontakte, Viber, можуть продавати інформацію компаніям, або просто акаунти можуть взламуватися.

Під час виконання випускної роботи було використано об’єктно-орієнтовану мову програмування Java та такі реляційні бази даних як MySQL та SQLite. Було використано сервер повідомлень RabbitMQ. Були поглиблені знання з методів та підходів для шифрування даних. Зокрема, було досконально вивчено механізм RSA AES session key.

СПИСОК ЛІТЕРАТУРИ

1. “Миттєві повідомлення” [Електронний ресурс] Режим доступу: https://uk.wikipedia.org/wiki/Миттєві_повідомлення

2. “Названы самые популярные мессенджеры в Украине” [Електронний ресурс] Режим доступу: https://delo.ua/lifestyle/nazvany-samye-populjarnye-messendzhery-v-ukraine-360645

3. “Плюсы и минусы Viber” [Електронний ресурс] Режим доступу: https://sravni.cc/reviews/plyusy-i-minusy-viber

4. “Плюсы и минусы Facebook Messenger” [Електронний ресурс] Режим доступу: https://sravni.cc/reviews/plyusy-i-minusy-facebook-messenger

5. “Плюсы и минусы Телеграм” [Електронний ресурс] Режим доступу: https://sravni.cc/reviews/plyusy-i-minusy-telegram

6. “Top 10 In-Demand programming languages to learn in 2020” [Електронний ресурс] Режим доступу: https://towardsdatascience.com/top-10-in-demand-programming-languages-to-learn-in-2020-4462eb7d8d3e

7. “Обзор основных языков программирования” [Електронний ресурс] Режим доступу: https://www.13min.ru/it/obzor-osnovnyx-yazykov-programmirovaniya

8. “C# — Преимущества и недостатки” [Електронний ресурс] Режим доступу: https://shwanoff.ru/plus-minus-c-sharp

9. “SQLite vs MySQL vs PostgreSQL: сравнение систем управления базами данных” [Електронний ресурс] Режим доступу: https://devacademy.ru/article/sqlite-vs-mysql-vs-postgresql

10. “Плюсы и минусы программирования на Java” [Електронний ресурс] Режим доступу: https://medium.com/nuances-of-programming/плюсы-и-минусы-программирования-на-java-2861f4c2a0d5

11. “MySQL: особенности и сферы применения” [Електронний ресурс] Режим доступу: https://www.bytemag.ru/articles/detail.php?ID=6547

12. “SQLite” [Електронний ресурс] Режим доступу: https://ru.wikipedia.org/wiki/SQLite

13. “Java Database Connectivity” [Електронний ресурс] Режим доступу: https://uk.wikipedia.org/wiki/Java_Database_Connectivity

14. “MD5” [Електронний ресурс] Режим доступу: https://ru.wikipedia.org/wiki/MD5

15. “The MD5 Message-Digest Algorithm” [Електронний ресурс] Режим доступу: https://tools.ietf.org/html/rfc1321

16. “RSA” [Електронний ресурс] Режим доступу: https://ru.wikipedia.org/wiki/RSA

17. “RabbitMQ” [Електронний ресурс] Режим доступу: https://uk.wikipedia.org/wiki/RabbitMQ

18. “[Часть 1] RabbitMQ – Что такое RabbitMQ?” [Електронний ресурс] Режим доступу: https://thewebland.net/development/devops/what-is-rabbitmq

19. “RabbitMQ для начинающих” [Електронний ресурс] Режим доступу: http://ajaxblog.ru/php/rabbitmq-tutorial

20. Герберт Ш. Java. Полное руководство. 8-е издание - СПб.: Москва, 2017. - 1140 с.

21. “Public-Key Cryptography Standards” [Електронний ресурс] Режим доступу: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.402.6882&rep=rep1&type=pdf

ДОДАТОК А

Назви пакетів (package) та опис їх призначення у програмній реалізації.

Назва пакету	Призначення пакету
nameserver	**Кореневий пакет серверу імен**
dao	DAO для роботи з БД
dao/util	Допоміжні класи для валідації даних
domain	Головна частина серверу імен
domain/server	Класи логіки серверу імен
domain/settings	Класи завантаження налаштувань
exception	Класи виключень
model	POJO-моделі
dataserver	**Кореневий пакет серверу даних**
dao	DAO для роботи з БД
dao/util	Допоміжні класи для валідації даних
domain/server	Головна частина серверу даних
domain/settings	Класи завантаження налаштувань
exception	Класи виключень
model	POJO-моделі
protocol	**Кореневий пакет протоколу**
main/java/dnm/protocol	Кореневий пакет бібліотек протоколу
main/java/dnm/protocol/crypto	Класи для шифрування та дешифрування даних, обрахунку контрольних сум
main/java/dnm/protocol/database	Інтерфейси необхідних класів для роботи з БД
main/java/dnm/protocol/database/model	Моделі для інтерфейсів

main/java/dnm/protocol/exception	Класи виключень
main/java/dnm/protocol/message	Класи повідомлення
main/java/dnm/protocol/message/util	Допоміжні класи
main/java/dnm/protocol/processor	Пакети та класи обробників різних типів повідомлень
main/java/dnm/protocol/processor/dataserver	Обробники повідомлень для серверу даних
main/java/dnm/protocol/processor/factory	Фабрика обробників
main/java/dnm/protocol/processor/nameserver	Обробники повідомлень для серверу імен
main/java/dnm/protocol/processor/validator	Валідатор різних типів повідомлень
test/java/dnm/protocol	Юніт-тести протоколу
test/java/dnm/protocol/crypto	Юніт-тести крипто частини протоколу
test/java/dnm/protocol/message/util	Юніт-тести допоміжних класів протоколу
test/java/dnm/protocol/processor	Юніт-тести обробників
test/java/dnm/protocol/processor/nameserver	Юніт-тести обробників серверу імен
test/java/dnm/protocol/processor/validator	Юніт-тести валідаторів
client	**Кореневий пакет клієнта**
java/dnm/client	Класи контроллерів форм
java/dnm/client/controllers	Класи виключень
java/dnm/client/exceptions	Класи для роботи з мережею

java/dnm/client/models	POJO-моделі для роботи з локальною БД налаштувань
java/dnm/client/models/viewable	POJO-моделі для відображення у формах JavaFX
java/dnm/client/settings	Класи для завантаження та збереження налаштувань
java/dnm/client/settings/dao	DAO для роботи з локальною БД налаштувань
resources	Пакет ресурсів програми
resources/icons	Зображення та іконки
resources/sounds	Звукові файли
resources/views	FXML код форм інтерфейсу

Далі приведено програмний код додатку.

ДОДАТОК Б

Dataserver, package dao:

dao/util/ModelValidator.java

```
package dnm.dataserver.dao.util;

public class ModelValidator {
  public static boolean anyIsNull(Object... parameters) {
    for (Object parameter: parameters) {
      if (parameter == null) {
        return true;
      }
    }
    return false;
  }

  public static boolean anyIsEmpty(String... parameters) {
    for (String parameter: parameters) {
      if (parameter.isEmpty()) {
        return true;
      }
    }
    return false;
  }
}
```

dao/DataServerDao.java

```
package dnm.dataserver.dao;

import dnm.dataserver.model.DataServer;

import javax.sql.DataSource;
import java.sql.*;
import java.util.logging.Logger;

import static dnm.dataserver.dao.util.ModelValidator.*;

public class DataServerDao {
  private static final Logger log = Logger.getLogger(DataServerDao.class.getName());

  private final DataSource dataSource;

  public DataServerDao(DataSource dataSource) {
    this.dataSource = dataSource;
  }

  public void createDataServer(String address, String publicKey, String privateKey) {
    DataServer existingDataServer = getDataServer();
    if (!isNullOrEmpty(existingDataServer)) {
      log.warning("Trying to add another data server information, using update method instead.");
      updateDataServer(new DataServer(address, publicKey, privateKey));
      return;
    }

    String query = "INSERT INTO dataserver (address, pubkey, privkey) VALUES (?, ?, ?)";
    try (Connection connection = dataSource.getConnection();
       PreparedStatement statement = connection.prepareStatement(query)) {
      connection.setAutoCommit(false);

      statement.setString(1, address);
      statement.setString(2, publicKey);
      statement.setString(3, privateKey);

      int rowsAffected = statement.executeUpdate();
      if (rowsAffected == 0) {
        log.severe("Error adding data server info! No rows affected!");
        throw new SQLException("Cannot add data server info, no rows affected.");
      }

      connection.commit();
```

```
    log.info(String.format("data server info added: %s.", address));
  } catch (SQLException e) {
    log.severe(String.format("Error happened during adding data server info: %s.", e.getMessage()));
  }
}

public void createDataServer(DataServer dataServer) {
  if (isNullOrEmpty(dataServer)) {
    log.severe("Tried to add name server information with empty or null data.");
    return;
  }
  createDataServer(dataServer.getAddress(), dataServer.getPublicKey(), dataServer.getPrivateKey());
}

public DataServer getDataServer() {
  DataServer DataServer = null;

  String query = "SELECT address, pubkey, privkey FROM DataServer";
  try (Connection connection = dataSource.getConnection();
     Statement statement = connection.createStatement();
     ResultSet result = statement.executeQuery(query)) {
    if (result.first()) {
      DataServer = new DataServer();
      DataServer.setAddress(result.getString("address"));
      DataServer.setPublicKey(result.getString("pubkey"));
      DataServer.setPrivateKey(result.getString("privkey"));
    }
  } catch (SQLException e) {
    log.severe(String.format("Error happened during getting data server info: %s.", e.getMessage()));
  }

  return DataServer;
}

public void updateDataServer(DataServer DataServer) {
  DataServer existingDataServer = getDataServer();
  if (isNullOrEmpty(existingDataServer)) {
    createDataServer(DataServer);
    return;
  }

  if (existingDataServer.equals(DataServer)) {
    log.warning("Updating data server info with the same as it was earlier operation omitted.");
    return;
  }

  String query = "UPDATE DataServer SET address=?, pubkey=?, privkey=?";
  try (Connection connection = dataSource.getConnection();
     PreparedStatement statement = connection.prepareStatement(query)) {
    connection.setAutoCommit(false);

    statement.setString(1, DataServer.getAddress());
    statement.setString(2, DataServer.getPublicKey());
    statement.setString(3, DataServer.getPrivateKey());

    int rowsAffected = statement.executeUpdate();
    if (rowsAffected == 0) {
      log.severe("Error updating data server info! No rows affected!");
      throw new SQLException("Cannot add data server info, no rows affected.");
    }

    connection.commit();

    log.info("data server info has been updated.");
  } catch (SQLException e) {
    log.severe(String.format("Error happened during updating data server info: %s.", e.getMessage()));
  }
}

public void updateDataServerAddress(String address) {
  DataServer existingDataServer = getDataServer();
  if (isNullOrEmpty(existingDataServer)) {
    log.severe(String.format("Cannot update data server address (%s), because no data server info exists.",
        address));
    return; // todo: exception
  }
```

```
    String oldAddress = existingDataServer.getAddress();
    if (oldAddress.equals(address)) {
      log.warning(String.format("Setting up the same address (%s) as it wes operation omitted.", address));
      return;
    }

    existingDataServer.setAddress(address);
    updateDataServer(existingDataServer);
  }

  public void updateDataServerKeys(String publicKey, String privateKey) {
    DataServer existingDataServer = getDataServer();
    if (isNullOrEmpty(existingDataServer)) {
      log.severe("Cannot update data server keys, because no data server info exists.");
      return; // todo: exception
    }

    String oldPublicKey = existingDataServer.getPublicKey();
    String oldPrivateKey = existingDataServer.getPrivateKey();
    if (oldPublicKey.equals(privateKey) && oldPrivateKey.equals(privateKey)) {
      log.warning("Setting up the same keys as they were operation omitted.");
      return;
    }

    existingDataServer.setPublicKey(publicKey);
    existingDataServer.setPrivateKey(privateKey);
    updateDataServer(existingDataServer);
  }

  // no input args, because table consists of only one entry.
  public void removeDataServer() {
    DataServer existingDataServer = getDataServer();
    if (isNullOrEmpty(existingDataServer)) {
      log.severe("Cannot remove not existing data server info.");
      return;
    }

    String query = "DELETE FROM dataserver";
    try (Connection connection = dataSource.getConnection();
       Statement statement = connection.createStatement()) {
      connection.setAutoCommit(false);
      statement.executeQuery(query);
      connection.commit();
      log.info("data server information removed.");
    } catch (SQLException e) {
      log.severe(String.format("Error happened during removing data server info: %s.", e.getMessage()));
    }
  }

  private boolean isNullOrEmpty(DataServer dataServer) {
    if (anyIsNull(dataServer)) {
      return true;
    }

    String address = dataServer.getAddress();
    String publicKey = dataServer.getPublicKey();
    String privateKey = dataServer.getPrivateKey();

    return anyIsNull(address, publicKey, privateKey) || anyIsEmpty(address, publicKey, privateKey);
  }
}
```

dao/MessageDao.java

```
package dnm.dataserver.dao;

import dnm.dataserver.model.Message;

import javax.sql.DataSource;
import java.sql.*;
import java.util.*;
import java.util.logging.Logger;
import java.util.stream.Collectors;

import static dnm.dataserver.dao.util.ModelValidator.*;
```

```
public class MessageDao {
  public static final int MAX_MESSAGES_NUMBER = 50;
  public static final int MAX_MESSAGE_SYMBOLS_NUMBER = 4128;

  private static final Logger log = Logger.getLogger(MessageDao.class.getName());

  private final DataSource dataSource;

  public MessageDao(DataSource dataSource) {
    this.dataSource = dataSource;
  }

  public void createMessage(int id, int chatId, int authorId, String text) {
    int messageTextSize = Objects.requireNonNull(text, "Message text cannot be null.").length();
    if (messageTextSize <= MAX_MESSAGE_SYMBOLS_NUMBER) {
      createValidatedMessages(List.of(new Message(id, chatId, authorId, text)));
    } else {
      log.severe(String.format("Trying to save message with size larger than should be: %d > %d.",
          messageTextSize, MAX_MESSAGE_SYMBOLS_NUMBER));
    }
  }

  public void createMessage(Message message) {
    if (isNullOrEmpty(message)) {
      log.warning("Trying to create message with null or empty information.");
      return;
    }
    createMessage(message.getId(), message.getChatId(), message.getAuthorId(), message.getText());
  }

  public void createMessages(List<Message> messages) {
    List<Message> validatedMessages = Objects.requireNonNull(messages)
        .stream()
        .filter((Message message) -> !isNullOrEmpty(message))
        .collect(Collectors.toList());
    createValidatedMessages(validatedMessages);
  }

  public Message getMessage(int id, int chatId) {
    Message queriedMessage = null;
    String query = "SELECT id, chat_id, author_id, text FROM messages WHERE id=? AND chat_id=?";
    try (Connection connection = dataSource.getConnection();
      PreparedStatement statement = connection.prepareStatement(query)) {
      connection.setAutoCommit(false);

      statement.setInt(1, id);
      statement.setInt(2, chatId);

      try (ResultSet result = statement.executeQuery()) {
        if (result.first()) {
          queriedMessage = new Message();
          queriedMessage.setId(result.getInt("id"));
          queriedMessage.setChatId(result.getInt("chat_id"));
          queriedMessage.setAuthorId(result.getInt("author_id"));
          queriedMessage.setText(result.getString("text"));

          log.fine(String.format("Successfully got message from chat %d by id %d.", chatId, id));
        } else {
          log.warning("Error getting message from database.");
        }
      }

      connection.commit();
    } catch (SQLException e) {
      log.severe(String.format("Error happened during getting a message: %s.", e.getMessage()));
    }
    return queriedMessage;
  }

  public List<Message> getMessages(int chatId, int offset, int count) {
    if (count > MAX_MESSAGES_NUMBER) {
      log.warning("Trying to fetch more messages than allowed. Using maximum allowed number.");
      count = MAX_MESSAGES_NUMBER;
    }

    List<Message> queriedMessages = new ArrayList<>(count);
    String query = "SELECT id, chat_id, author_id, text FROM messages WHERE chat_id=? ORDER BY id DESC LIMIT
```

```
? OFFSET ?";
    try (Connection connection = dataSource.getConnection();
         PreparedStatement statement = connection.prepareStatement(query)) {
      connection.setAutoCommit(false);

      statement.setInt(1, chatId);
      statement.setInt(2, count);
      statement.setInt(3, offset);

      try (ResultSet result = statement.executeQuery()) {
        while (result.next()) {
          Message message = new Message();
          message.setId(result.getInt("id"));
          message.setChatId(chatId);
          message.setText(result.getString("text"));
          queriedMessages.add(message);
        }
      }

      connection.commit();
    } catch (SQLException e) {
      log.severe(String.format("Error happened during getting messages: %s.", e.getMessage()));
    }

    return queriedMessages;
  }

  public void updateMessage(Message message) {
    Message existingMessage = getMessage(message.getId(), message.getChatId());
    if (isNullOrEmpty(existingMessage)) {
      log.warning(String.format("Trying to update not existing message: %d %d.",
          message.getId(), message.getChatId()));
      return;
    }

    if (existingMessage.equals(message)) {
      log.warning("Updating message with the same data operation omitted.");
      return;
    }

    String query = "UPDATE messages SET text=? WHERE id=? AND chat_id=?";
    try (Connection connection = dataSource.getConnection();
         PreparedStatement statement = connection.prepareStatement(query)) {
      connection.setAutoCommit(false);

      statement.setString(1, message.getText());
      statement.setInt(2, message.getId());
      statement.setInt(3, message.getChatId());

      int rowsAffected = statement.executeUpdate();
      if (rowsAffected == 0) {
        log.severe("Error happened during updating message: no rows affected.");
      } else {
        log.fine("Message successfully updated.");
      }

      connection.commit();
    } catch (SQLException e) {
      log.severe(String.format("Error happened during updating message: %s.", e.getMessage()));
    }
  }

  public void deleteMessage(int id, int chatId) {
    Message messageToDelete = getMessage(id, chatId);
    if (anyIsNull(messageToDelete)) {
      log.warning("Trying to delete not existing message!");
      return;
    }

    String query = "DELETE FROM messages WHERE id=? AND chat_id=?";
    try (Connection connection = dataSource.getConnection();
         PreparedStatement statement = connection.prepareStatement(query)) {
      connection.setAutoCommit(false);

      statement.setInt(1, id);
      statement.setInt(2, chatId);
      int rowsAffected = statement.executeUpdate();
```

```
      if (rowsAffected == 0) {
        log.severe("Error deleting message: no rows affected.");
      } else {
        log.fine("Message successfully deleted.");
      }

      connection.commit();
    } catch (SQLException e) {
      log.severe(String.format("Error happened during deleting the message: %s.", e.getMessage()));
    }
  }

  private void createValidatedMessages(List<Message> messages) {
    String query = "INSERT INTO messages (id, chat_id, author_id, text) VALUES (?, ?, ?, ?)";
    try (Connection connection = dataSource.getConnection();
        PreparedStatement statement = connection.prepareStatement(query)) {
      connection.setAutoCommit(false);

      for (Message message : messages) {
        statement.setInt(1, message.getId());
        statement.setInt(2, message.getChatId());
        statement.setInt(3, message.getAuthorId());
        statement.setString(4, message.getText());
        statement.addBatch();
      }

      int[] rowsAffectedArray = statement.executeBatch();
      for (int rowsAffected : rowsAffectedArray) {
        if (rowsAffected == 0) {
          log.severe("Error happened during adding new messages to database: no rows affected!");
        } else {
          log.fine("Created new messages.");
        }
      }

      connection.commit();
    } catch (SQLException e) {
      log.severe(String.format("Error happened during creating a message: %s.", e.getMessage()));
    }
  }

  private boolean isNullOrEmpty(Message message) {
    if (anyIsNull(message)) {
      return true;
    }

    String text = message.getText();

    return anyIsNull(text) || anyIsEmpty(text);
  }
}
```

package domain:

domain/server/DataServer.java

```
package dnm.dataserver.domain.server;

import com.rabbitmq.client.*;
import dnm.dataserver.domain.settings.SettingsRepository;
import dnm.protocol.Protocol;
import dnm.protocol.exception.MessageException;
import dnm.protocol.message.ErrorMessageType;
import dnm.protocol.message.Message;
import dnm.protocol.message.util.MessageSerializerDeserializer;
import dnm.protocol.message.util.Messages;

import java.io.IOException;
import java.util.concurrent.TimeoutException;
import java.util.logging.Logger;

public class DataServer {
  private static final String DARASERVER_QUEUE_NAME = "dataserver_queue";

  private static final Logger log = Logger.getLogger(DataServer.class.getName());
```

```
  private final SettingsRepository settings;
  private final Protocol protocol;

  public DataServer(SettingsRepository settings, Protocol protocol) {
    this.settings = settings;
    this.protocol = protocol;

    log.setLevel(settings.getLoggingLevel());
  }

  private Message getResponse(Delivery delivery) {
    Message response;
    try {
      byte[] rawMessage = delivery.getBody();
      Message message = new MessageSerializerDeserializer(Messages.fromRawMessage(rawMessage));
      response = new MessageSerializerDeserializer(protocol.processMessage(message));
    } catch (RuntimeException e) {
      System.out.println(" [.] " + e.toString());
      response = Messages.getErrorMessage(ErrorMessageType.CANNOT_PROCESS);
    } catch (MessageException e) {
      e.printStackTrace();
      response = Messages.getErrorMessage(ErrorMessageType.CANNOT_PROCESS);
    }
    return response;
  }

  public void run() {
    String hostname = settings.getServerHostname();
    int port = settings.getServerPortNumber();

    ConnectionFactory connectionFactory = new ConnectionFactory();
    connectionFactory.setHost(hostname);
    connectionFactory.setPort(port);

    try (Connection connection = connectionFactory.newConnection();
       Channel channel = connection.createChannel()) {
      channel.queueDeclare(DARASERVER_QUEUE_NAME, false, false, false, null);
      channel.queuePurge(DARASERVER_QUEUE_NAME);

      channel.basicQos(1);

      log.info(String.format("Server is waiting for messages on %s:%d. Q name = %s.",
          hostname, port, DARASERVER_QUEUE_NAME));

      Object monitor = new Object();

      DeliverCallback deliverCallback = (consumerTag, delivery) -> {
        AMQP.BasicProperties replyProperties = new AMQP.BasicProperties.Builder()
            .correlationId(delivery.getProperties().getCorrelationId())
            .build();

        Message response = getResponse(delivery);
        channel.basicPublish("", delivery.getProperties().getReplyTo(), replyProperties,
Messages.toRawMessage(response));
        channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);

        synchronized (monitor) {
          monitor.notify();
        }
      };

      channel.basicConsume(DARASERVER_QUEUE_NAME, false, deliverCallback, (consumerTag -> { }));

      while (true) {
        synchronized (monitor) {
          try {
            monitor.wait();
          } catch (InterruptedException e) {
            e.printStackTrace();
          }
        }
      }
    } catch (TimeoutException | IOException e) {
      log.severe(String.format("Error happened while running data server: %s.", e.getMessage()));
    }
```

```
  }
}
```

domain/server/DataServerMessageSender.java

```
package dnm.dataserver.domain.server;

import dnm.protocol.MessageSender;
import dnm.protocol.message.Message;
import dnm.protocol.message.util.Messages;

import java.util.logging.Logger;

import static dnm.protocol.message.ErrorMessageType.CANNOT_PROCESS;

public class DataServerMessageSender implements MessageSender {
  private static final Logger log = Logger.getLogger(DataServerMessageSender.class.getName());

  @Override
  public Message send(Message message) {
    log.severe("Dataserver message sender should not be called! Check protocol version. Returning
CANNOT_PROCESS.");
    return Messages.getErrorMessage(CANNOT_PROCESS);
  }
}
```

domain/settings/JsonSettingsRepository.java

```
package dnm.dataserver.domain.settings;

import com.google.gson.Gson;
import com.google.gson.JsonSyntaxException;
import com.google.gson.reflect.TypeToken;
import dnm.dataserver.exception.SettingsRepositoryException;

import java.io.*;
import java.lang.reflect.Type;
import java.util.HashMap;
import java.util.Map;
import java.util.Objects;
import java.util.logging.Level;
import java.util.logging.Logger;

public class JsonSettingsRepository implements SettingsRepository {
  private static final Logger log = Logger.getLogger(JsonSettingsRepository.class.getName());

  private final Gson json = new Gson();

  private File settingsFile;

  private Level loggingLevel = Level.INFO;

  private String dbHostname = "localhost";
  private int dbPortNumber = 3306;
  private String dbUsername = "admin";
  private String dbPassword = "qwerty";

  private String serverHostname = "localhost";
  private int serverPortNumber = 5647;

  public JsonSettingsRepository(String filename) {
    this.settingsFile = new File(filename);
  }

  @Override
  public void loadFromFile() throws SettingsRepositoryException {
    try (InputStream fileInputStream = new FileInputStream(settingsFile)) {
      String data = new String(fileInputStream.readAllBytes());
      this.fromMap(parseSettings(data));
      log.fine(String.format("Successfully loaded setting from file %s.", getFilename()));
    } catch (FileNotFoundException e) {
      log.warning(String.format("Trying to load setting from not existing file: %s.", getFilename()));
      throw new SettingsRepositoryException(String.format("Error loading from not existing file %s!",
          getFilename()));
    } catch (IOException e) {
      log.severe(String.format("Error happened during working with settings file: %s.", e.getMessage()));
```

```
      throw new SettingsRepositoryException("Error working with settings file.");
    }
  }

  @Override
  public void saveToFile() throws SettingsRepositoryException {
    try (OutputStream fileOutputStream = new FileOutputStream(settingsFile)) {
      fileOutputStream.write(dumpSettings().getBytes());
      fileOutputStream.flush();
      log.fine(String.format("Successfully saved setting to file %s.", getFilename()));
    } catch (FileNotFoundException e) {
      log.warning(String.format("Trying to save settings to not existing file: %s.", getFilename()));
      throw new SettingsRepositoryException(String.format("Error saving to not existing file %s!",
          getFilename()));
    } catch (IOException e) {
      log.severe(String.format("Error happened during working with settings file: %s.", e.getMessage()));
      throw new SettingsRepositoryException("Error working with settings file.");
    }
  }

  @Override
  public String getFilename() {
    return settingsFile.getName();
  }

  @Override
  public void setFilename(String filename) {
    this.settingsFile = new File(filename);
  }

  @Override
  public String getDbHostname() {
    return dbHostname;
  }

  @Override
  public void setDbHostname(String dbHostname) {
    this.dbHostname = dbHostname;
  }

  @Override
  public int getDbPortNumber() {
    return this.dbPortNumber;
  }

  @Override
  public void setDbPortNumber(int portNumber) {
    this.dbPortNumber = portNumber;
  }

  @Override
  public String getDbUsername() {
    return dbUsername;
  }

  @Override
  public void setDbUsername(String dbUsername) {
    this.dbUsername = dbUsername;
  }

  @Override
  public String getDbPassword() {
    return dbPassword;
  }

  @Override
  public void setDbPassword(String dbPassword) {
    this.dbPassword = dbPassword;
  }

  @Override
  public Level getLoggingLevel() {
    return loggingLevel;
  }

  @Override
  public void setLoggingLevel(Level loggingLevel) {
```

```
    this.loggingLevel = loggingLevel;
  }

  @Override
  public String getServerHostname() {
    return serverHostname;
  }

  @Override
  public void setServerHostname(String serverHostname) {
    this.serverHostname = serverHostname;
  }

  @Override
  public int getServerPortNumber() {
    return serverPortNumber;
  }

  @Override
  public void setServerPortNumber(int serverPortNumber) {
    this.serverPortNumber = serverPortNumber;
  }

  private Map<String, Object> toMap() {
    Map<String, String> dbSettings = new HashMap<>();
    dbSettings.put("hostname", getDbHostname());
    dbSettings.put("port", String.valueOf(getDbPortNumber()));
    dbSettings.put("username", getDbUsername());
    dbSettings.put("password", getDbPassword());

    Map<String, String> serverSettings = new HashMap<>();
    serverSettings.put("hostname", getServerHostname());
    serverSettings.put("port", String.valueOf(getServerPortNumber()));

    Map<String, Object> settings = new HashMap<>();
    settings.put("log_level", getLoggingLevel());
    settings.put("db", dbSettings);
    settings.put("server", serverSettings);

    return settings;
  }

  @SuppressWarnings("unchecked")
  private void fromMap(Map<String, Object> settings) {
    this.setLoggingLevel(Level.parse((String) settings.getOrDefault("log_level", "INFO")));

    Map<String, String> dbSettings = (Map<String, String>) settings.getOrDefault("db", new HashMap<>());
    this.setDbHostname(dbSettings.getOrDefault("hostname", "localhost"));
    this.setDbPortNumber(Integer.parseInt(dbSettings.getOrDefault("port", "3306")));
    this.setDbUsername(dbSettings.getOrDefault("username", "root"));
    this.setDbPassword(dbSettings.getOrDefault("password", "test"));

    Map<String, String> serverSettings = (Map<String, String>) settings.getOrDefault("server", new
HashMap<>());
    this.setServerHostname(serverSettings.getOrDefault("hostname", "localhost"));
    this.setServerPortNumber(Integer.parseInt(serverSettings.getOrDefault("port", "5647")));
  }

  private Map<String, Object> parseSettings(String rawSettings) {
    Type settingsType = new TypeToken<Map<String, Object>>() { }.getType();
    Map<String, Object> settings = new HashMap<>();
    try {
      settings = json.fromJson(rawSettings, settingsType);
    } catch (JsonSyntaxException e) {
      log.warning("Cannot deserialize settings from raw type, wrong structure. Using default settings
instead.");
    }
    return settings;
  }

  private String dumpSettings() {
    return json.toJson(Objects.requireNonNullElse(toMap(), ""));
  }
}
```

domain/settings/SettingsRepository.java

```
package dnm.dataserver.domain.settings;

import dnm.dataserver.exception.SettingsRepositoryException;

import java.util.logging.Level;

public interface SettingsRepository {
  void loadFromFile() throws SettingsRepositoryException;
  void saveToFile() throws SettingsRepositoryException;

  String getFilename();
  void setFilename(String filename);

  Level getLoggingLevel();
  void setLoggingLevel(Level level);

  String getDbHostname();
  void setDbHostname(String hostname);

  int getDbPortNumber();
  void setDbPortNumber(int portNumber);

  String getDbUsername();
  void setDbUsername(String username);

  String getDbPassword();
  void setDbPassword(String password);

  String getServerHostname();
  void setServerHostname(String serverHostname);

  int getServerPortNumber();
  void setServerPortNumber(int serverPortNumber);
}
```

domain/DataServerDatabaseImpl.java

```
package dnm.dataserver.domain;

import com.mysql.cj.jdbc.MysqlDataSource;

import dnm.dataserver.dao.DataServerDao;
import dnm.dataserver.dao.MessageDao;
import dnm.dataserver.domain.settings.SettingsRepository;
import dnm.dataserver.model.DataServer;
import dnm.dataserver.model.Message;
import dnm.protocol.crypto.KeyPair;
import dnm.protocol.crypto.KeyPairGenerator;
import dnm.protocol.crypto.RsaEncryptorDecryptor;
import dnm.protocol.database.DataServerDatabase;
import dnm.protocol.database.model.MessageInfo;

import javax.sql.DataSource;
import java.sql.SQLException;
import java.util.List;
import java.util.Objects;
import java.util.logging.Logger;
import java.util.stream.Collectors;

import static jdk.jshell.spi.ExecutionControl.*;

public class DataServerDatabaseImpl implements DataServerDatabase {
  private static final int MAX_MESSAGES_NUMBER = MessageDao.MAX_MESSAGES_NUMBER;
  private static final int MAX_MESSAGE_SYMBOLS_NUMBER = MessageDao.MAX_MESSAGE_SYMBOLS_NUMBER;

  private static final Logger log = Logger.getLogger(DataServerDatabaseImpl.class.getName());

  private SettingsRepository settings;

  private final DataServerDao dataServerDao;
  private final MessageDao messageDao;


  public DataServerDatabaseImpl(SettingsRepository settings) {
    this.settings = settings;

    DataSource dataSource = prepareDataSource();
```

```
  dataServerDao = new DataServerDao(dataSource);
  messageDao = new MessageDao(dataSource);
}

private DataSource prepareDataSource() {
  String url = String.format("jdbc:mysql://%s:%d/dataserver_schema", settings.getDbHostname(),
      settings.getDbPortNumber());

  MysqlDataSource dataSource = new MysqlDataSource();
  dataSource.setUrl(url);
  dataSource.setUser(settings.getDbUsername());
  dataSource.setPassword(settings.getDbPassword());

  try {
    dataSource.setServerTimezone("UTC");
  } catch (SQLException e) {
    log.severe(String.format("Error happened during setting server timezone: %s.", e.getMessage()));
  }

  log.fine("Data source successfully created!");

  return dataSource;
}

private void createNameServer() {
  log.warning("Data server record isn't created yet.");

  DataServer nameServer = new DataServer();
  nameServer.setAddress(settings.getServerHostname());

  KeyPairGenerator rsa = new RsaEncryptorDecryptor();
  KeyPair keys = rsa.generateKeyPair();

  nameServer.setPublicKey(keys.getPublicKey());
  nameServer.setPrivateKey(keys.getPrivateKey());

  dataServerDao.createDataServer(nameServer);

  nameServer = dataServerDao.getDataServer();
  if (Objects.nonNull(nameServer)) {
    log.info("Created new data server info record in database.");
  } else {
    log.warning("Error creating data server, check db connection!");
  }
}

/**
 * actually not designed but may be needed later.
 * */
public String getDataServerAddress() throws NotImplementedException {
  DataServer dataServer = dataServerDao.getDataServer();
  if (Objects.isNull(dataServer)) {
    createNameServer();
    return settings.getServerHostname();
  }
  return dataServer.getAddress();
}

@Override
public String getPublicKey() {
  DataServer dataServer = dataServerDao.getDataServer();
  if (Objects.isNull(dataServer)) {
    createNameServer();
    return null;
  }
  return dataServer.getPublicKey();
}

@Override
public byte[] getPrivateKey() {
  DataServer dataServer = dataServerDao.getDataServer();
  if (Objects.isNull(dataServer)) {
    createNameServer();
    return null;
  }
  return dataServer.getPrivateKey().getBytes();
```

```
  }

  @Override
  public boolean saveMessage(int messageId, int chatId, int authorId, String text) {
    if (messageId == -1 || chatId == -1 || authorId == -1 || Objects.isNull(text)
        || text.isEmpty() || text.isBlank() || text.length() > MAX_MESSAGE_SYMBOLS_NUMBER) {
      log.info("Input parameters dont satisfy requirements.");
      return false;
    }

    messageDao.createMessage(messageId, chatId, authorId, text);

    Message createdMessage = messageDao.getMessage(messageId, chatId);
    if (Objects.isNull(createdMessage)) {
      log.severe(String.format("Error adding message (%d) to data server's database.", messageId));
      return false;
    }

    log.fine(String.format("Successfully added message %d:%d to data server's database!", chatId,
messageId));

    return true;
  }

  @Override
  public MessageInfo getMessage(int id, int chatId) {
    return toMessageInfo(messageDao.getMessage(id, chatId));
  }

  @Override
  public List<MessageInfo> getMessages(int chatId, int offset, int count) {
    if (count > MAX_MESSAGES_NUMBER) {
      log.info(String.format("Trying to get more messages than allowed: %d > %d.", count,
MAX_MESSAGES_NUMBER));
      return null;
    }

    List<Message> queriedMessages = messageDao.getMessages(chatId, offset, count);
    if (Objects.isNull(queriedMessages)) {
      log.info(String.format("Cannot fetch %d messages with offset %d from db by chat id %d",
          count, offset, chatId));
      return null;
    }

    log.info(String.format("Fetched %d messages of %d with offset %d from chat %d.",
        queriedMessages.size(), count, offset, chatId));

    return queriedMessages.stream()
        .map(this::toMessageInfo)
        .collect(Collectors.toList());
  }

  @Override
  public byte[] getSessionKey(String sessionId) {
    log.severe("Trying to get session key from data server database, but this functional not implemented
yet!");
    return null;
  }

  private MessageInfo toMessageInfo(Message message) {
    int messageId = message.getId();
    int chatId = message.getChatId();
    int authorId = message.getAuthorId();
    String text = message.getText();

    MessageInfo messageInfo = new MessageInfo();
    messageInfo.setId(messageId);
    messageInfo.setChatId(chatId);
    messageInfo.setAuthorId(authorId);
    messageInfo.setText(text);

    return messageInfo;
  }
}
```

package exception:

exception/SettingsRepositoryException.java

```
package dnm.dataserver.exception;

public class SettingsRepositoryException extends Exception {
  public SettingsRepositoryException(String message) {
    super(message);
  }
}
```

package model:

model/DataServer.java

```
package dnm.dataserver.model;

import java.util.Objects;

public class DataServer {
  private String address;
  private String publicKey;
  private String privateKey;

  public DataServer() {

  }

  public DataServer(String address, String publicKey, String privateKey) {
    this.address = address;
    this.publicKey = publicKey;
    this.privateKey = privateKey;
  }

  public String getAddress() {
    return address;
  }

  public void setAddress(String address) {
    this.address = address;
  }

  public String getPublicKey() {
    return publicKey;
  }

  public void setPublicKey(String publicKey) {
    this.publicKey = publicKey;
  }

  public String getPrivateKey() {
    return privateKey;
  }

  public void setPrivateKey(String privateKey) {
    this.privateKey = privateKey;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    DataServer that = (DataServer) o;
    return getAddress().equals(that.getAddress()) &&
        getPublicKey().equals(that.getPublicKey()) &&
        getPrivateKey().equals(that.getPrivateKey());
  }

  @Override
  public int hashCode() {
    return Objects.hash(getAddress(), getPublicKey(), getPrivateKey());
  }
}
```

model/Message.java

```java
package dnm.dataserver.model;

import java.util.Objects;

public class Message {
  private int id;
  private int chatId;
  private int authorId;
  private String text;

  public Message() {

  }

  public Message(int id, int chatId, int authorId, String text) {
    this.id = id;
    this.chatId = chatId;
    this.authorId = authorId;
    this.text = text;
  }

  public int getId() {
    return id;
  }

  public void setId(int id) {
    this.id = id;
  }

  public int getChatId() {
    return chatId;
  }

  public void setChatId(int chatId) {
    this.chatId = chatId;
  }

  public String getText() {
    return text;
  }

  public void setText(String text) {
    this.text = text;
  }

  public int getAuthorId() {
    return authorId;
  }

  public void setAuthorId(int authorId) {
    this.authorId = authorId;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Message message = (Message) o;
    return getId() == message.getId() &&
        getChatId() == message.getChatId() &&
        getAuthorId() == message.getAuthorId() &&
        getText().equals(message.getText());
  }

  @Override
  public int hashCode() {
    return Objects.hash(getId(), getChatId(), getAuthorId(), getText());
  }
}
```

package network:

network/DataServerProtocol.java

```
package dnm.dataserver.network;

public class DataServerProtocol {
}
```

Main.java

```
package dnm.dataserver;

import dnm.dataserver.domain.server.DataServer;
import dnm.dataserver.domain.DataServerDatabaseImpl;
import dnm.dataserver.domain.server.DataServerMessageSender;
import dnm.dataserver.domain.settings.JsonSettingsRepository;
import dnm.dataserver.domain.settings.SettingsRepository;
import dnm.dataserver.exception.SettingsRepositoryException;

import dnm.protocol.Protocol;
import dnm.protocol.ServerProtocol;
import dnm.protocol.processor.factory.DataServerMessageProcessorFactory;

import java.util.logging.Logger;

public class Main {
  private static final Logger log = Logger.getLogger(Main.class.getName());

  public static void main(String[] args) {
    if (args.length != 2) {
      log.severe("Settings file is not specified.");
      return;
    }

    String settingsFilename = args[1];
    SettingsRepository settings = new JsonSettingsRepository(settingsFilename);

    try {
      settings.loadFromFile();
    } catch (SettingsRepositoryException e) {
      log.severe(String.format("Error happened during loading settings from file \"%s\": %s.",
          settingsFilename, e.getMessage()));
      return;
    }

    log.info(String.format("Settings have been loaded from file \"%s\".", settingsFilename));

    Protocol protocol = new ServerProtocol(
        new DataServerMessageProcessorFactory(),
        new DataServerDatabaseImpl(settings),
        new DataServerMessageSender());

    DataServer dataServer = new DataServer(settings, protocol);
    dataServer.run();

    try {
      settings.saveToFile();
    } catch (SettingsRepositoryException e) {
      log.severe(String.format("Error happened during saving setting to file \"%s\": %s",
          settingsFilename, e.getMessage()));
    }

    log.info(String.format("Setting have been saved to file \"%s\".", settingsFilename));
    log.fine("Goodbye!");
  }
}
```